

# **PROGRAMAÇÃO PARA IOS**

## **GUIA DA BIG NERD RANCH**

**CHRISTIAN KEUR, AARON HILLEGASS & JOE CONWAY**



## Programação para iOS: Guia da Big Nerd Ranch

by Christian Keur, Aaron Hillegass, and Joe Conway

Copyright © 2014 Big Nerd Ranch, LLC

Todos os direitos reservados. Impresso nos Estados Unidos da América. Esta publicação é protegida por direitos autorais, e deve-se obter autorização da editora antes de qualquer reprodução, armazenamento em sistema de busca ou transmissão não autorizada, em qualquer formato ou por qualquer meio, seja eletrônico, mecânico, fotocópia, gravação ou similares. Para informações sobre autorizações, entre em contato com:

Big Nerd Ranch, LLC 1989 College Ave NE Atlanta, GA 30317 (404) 478-9005 <http://www.bignerdranch.com/> book-comments@bignerdranch.com

O logotipo de chapéu com hélice é uma marca registrada da Big Nerd Ranch, Inc.

A distribuição mundial da edição em inglês deste livro é exclusiva da:

Pearson Technology Group 800 East 96th Street Indianapolis, IN 46240 USA <http://www.informit.com>

Os autores e editores declaram ter tomado todas as precauções ao escrever e imprimir este livro, mas ainda assim, não fazem garantias expressas ou implícitas e não se responsabilizam por erros e omissões. Nenhuma responsabilidade será assumida por danos incidentais ou consequenciais, relacionados ou decorrentes do uso das informações ou programas contidos neste livro.

App Store, Apple, Cocoa, Cocoa Touch, Finder, Instruments, iCloud, iPad, iPhone, iPod, iPod touch, iTunes, Keychain, Mac, Mac OS, Multi-Touch, Objective-C, OS X, Quartz, Retina, Safari e Xcode são marcas comerciais da Apple, Inc., registradas nos Estados Unidos e em outros países.

Muitas das designações usadas pelos fabricantes e vendedores para distinguir seus produtos são consideradas marcas comerciais. Quando aplicáveis neste livro, e sempre que o editor estivesse ciente da existência de registro de marca comercial, as designações foram impressas com letras maiúsculas ou em caixa alta.

Quarta edição. Fevereiro de 2014

ISBN-10 0321942051

ISBN-13 978-0321942050

# Acknowledgments

Embora nossos nomes apareçam na capa, muitas pessoas ajudaram a tornar este livro uma realidade. Gostaríamos de aproveitar esta oportunidade para agradecê-las.

- Os outros instrutores que ministram aulas no iOS Bootcamp nos forneceram inúmeras sugestões e correções. São eles: Brian Hardy, Mikey Ward, Owen Mathews, Juan Pablo Claude, Rod Strougo, Jonathan Blocksom, Fernando Rodriguez, Jay Campbell, Matt Matthias, Scott Ritchie, Pouria Almassi, Step Christopher, TJ Usiyan e Bolot Kerimbaev. Esses instrutores foram muitas vezes auxiliados por alunos na busca pelos erros do livro, portanto, nossos agradecimentos são devidos a todos os alunos que participam do iOS Bootcamp.
- Em publicações modernas, o papel do editor foi reduzido a tal grau, que a palavra “edição” parece insuficiente para descrever o que Susan Loper fez por nós. Susan estudou cada palavra, cada frase, cada capítulo que escrevemos à procura de uma melhor palavra, melhor frase, melhor capítulo. Se você achou prazeroso ler o que escrevemos, temos de agradecer à genialidade de Susan.
- Nossos revisores técnicos, Chris Morris, Jawwad Ahmad e Véronique Brossier, nos ajudaram a encontrar e corrigir falhas.
- Ellie Volckhausen fez o projeto da capa. (A foto é do suporte inferior do chassi de uma bicicleta.)
- Chris Loper, da IntelligentEnglish.com, fez o projeto e produziu a versão impressa e as versões para EPUB e Kindle.
- A incrível equipe da Pearson Technology Group nos orientou com muita paciência por todo o processo de publicação do livro.

O agradecimento final e mais importante vai para nossos alunos cujas dúvidas foram nossa inspiração para escrever este livro e cujas frustrações nos inspiraram a fazer com que ele fosse claro e compreensível.



# Table of Contents

|                                                          |      |
|----------------------------------------------------------|------|
| Introdução .....                                         | xiii |
| Pré-requisitos .....                                     | xiii |
| O que mudou na quarta edição? .....                      | xiii |
| Nossa filosofia de ensino .....                          | xiii |
| Como usar este livro .....                               | xiv  |
| Como este livro está organizado .....                    | xv   |
| Opções de estilo .....                                   | xvi  |
| Convenções tipográficas .....                            | xvi  |
| Hardware e software necessários .....                    | xvii |
| 1. Um aplicativo iOS simples .....                       | 1    |
| Criação de um projeto no Xcode .....                     | 1    |
| Modelo-Visão-Controlador .....                           | 4    |
| Projetando o Quiz .....                                  | 5    |
| Criação de um controlador de visão .....                 | 5    |
| Construção de uma interface .....                        | 7    |
| Criação de objetos de visão .....                        | 8    |
| Configuração de objetos de visão .....                   | 9    |
| Arquivos NIB .....                                       | 11   |
| Fazendo conexões .....                                   | 11   |
| Criação de objetos de modelo .....                       | 15   |
| Utilização de preenchimento de código .....              | 16   |
| Juntando tudo .....                                      | 17   |
| Implementação de métodos de ação .....                   | 17   |
| Colocando o controlador de visão na tela .....           | 18   |
| Execução no simulador .....                              | 19   |
| Implementação de um aplicativo .....                     | 20   |
| Ícones de aplicativo .....                               | 21   |
| Imagens de abertura .....                                | 22   |
| 2. Objective-C .....                                     | 25   |
| Objetos .....                                            | 25   |
| Utilização de instâncias .....                           | 26   |
| Criação de objetos .....                                 | 26   |
| Envio de mensagens .....                                 | 26   |
| Destrução de objetos .....                               | 27   |
| Iniciando o RandomItems .....                            | 28   |
| Criação e preenchimento de um array .....                | 30   |
| Iteração em um array .....                               | 31   |
| Strings de formatação .....                              | 32   |
| Criação de subclasses em uma classe do Objective-C ..... | 33   |
| Criação de uma subclasse NSObject .....                  | 34   |
| Variáveis de instância .....                             | 35   |
| Acesso a variáveis de instância .....                    | 36   |
| Métodos de classe vs. de instância .....                 | 40   |
| Sobrescrevendo métodos .....                             | 40   |
| Inicializadores .....                                    | 41   |
| Métodos de classe .....                                  | 47   |
| Teste de subclasse .....                                 | 48   |
| Mais informações sobre NSArray e NSMutableArray .....    | 49   |
| Exceções e seletores não reconhecidos .....              | 50   |
| Desafios .....                                           | 52   |
| Desafio de bronze: detecção de bug .....                 | 52   |
| Desafio de prata: outro inicializador .....              | 53   |
| Desafio de ouro: outra classe .....                      | 54   |
| Você está mais curioso? .....                            | 54   |
| Para os mais curiosos: nomes de classe .....             | 54   |

|                                                                             |     |
|-----------------------------------------------------------------------------|-----|
| Para os mais curiosos: #import e @import .....                              | 55  |
| 3. Gerenciamento de memória com ARC .....                                   | 57  |
| A pilha .....                                                               | 57  |
| O Heap .....                                                                | 57  |
| ARC e gerenciamento de memória .....                                        | 58  |
| Variáveis ponteiros e propriedade de objeto .....                           | 58  |
| Como objetos perdem proprietários .....                                     | 59  |
| Cadeias de propriedade .....                                                | 60  |
| Referências fortes e fracas .....                                           | 62  |
| Propriedades .....                                                          | 66  |
| Declaração de propriedades .....                                            | 66  |
| Atributos de propriedades .....                                             | 69  |
| Acessores personalizados com propriedades .....                             | 71  |
| Para os mais curiosos: síntese de propriedades .....                        | 72  |
| Para os mais curiosos: pool de liberação automática e história da ARC ..... | 73  |
| 4. Visões e a hierarquia de visões .....                                    | 75  |
| Fundamentos da visão .....                                                  | 75  |
| A hierarquia de visões .....                                                | 76  |
| Criação da subclasse UIView .....                                           | 77  |
| Visões e frames .....                                                       | 78  |
| Projeções personalizadas em drawRect: .....                                 | 82  |
| Projeção de um único círculo .....                                          | 84  |
| UIBezierPath .....                                                          | 84  |
| Utilização da documentação do desenvolvedor .....                           | 85  |
| Projeção de círculos concêntricos .....                                     | 89  |
| Mais documentação do desenvolvedor .....                                    | 92  |
| Desafio de bronze: desenhar uma imagem .....                                | 92  |
| Para os mais curiosos: Core Graphics .....                                  | 93  |
| Desafio de ouro: sombras e gradientes .....                                 | 95  |
| 5. Visões: redesenho e UIScrollView .....                                   | 97  |
| Loop de execução e redesenho de visões .....                                | 98  |
| Extensões de classe .....                                                   | 99  |
| Utilização de UIScrollView .....                                            | 100 |
| Arraste de tela e paginação .....                                           | 102 |
| 6. Controladores de visão .....                                             | 105 |
| Criação de subclasses UIViewController .....                                | 106 |
| A visão de um controlador de visão .....                                    | 106 |
| Criação de uma visão programaticamente .....                                | 107 |
| Configuração do controlador de visão raiz .....                             | 108 |
| Outra UIViewController .....                                                | 108 |
| Criação de uma visão no Interface Builder .....                             | 110 |
| UITabBarController .....                                                    | 114 |
| Itens da barra de guias .....                                               | 116 |
| Inicializadores do UIViewController .....                                   | 118 |
| Adição de uma notificação local .....                                       | 119 |
| Visões carregadas e exibidas .....                                          | 119 |
| Acesso a subvisões .....                                                    | 120 |
| Interação com controladores de visão e suas visões .....                    | 121 |
| Desafio de bronze: outra guia .....                                         | 121 |
| Desafio de prata: lógica de controlador .....                               | 121 |
| Para os mais curiosos: codificação de chave-valor .....                     | 121 |
| Para os mais curiosos: tela Retina .....                                    | 122 |
| 7. Delegação e entrada de texto .....                                       | 125 |
| Campos de texto .....                                                       | 125 |
| UIResponder .....                                                           | 126 |
| Configuração do teclado .....                                               | 126 |
| Delegação .....                                                             | 128 |
| Protocolos .....                                                            | 129 |

|                                                                      |     |
|----------------------------------------------------------------------|-----|
| Adição de rótulos à tela .....                                       | 131 |
| Efeitos de movimentação .....                                        | 132 |
| Utilização do depurador .....                                        | 132 |
| Utilização de pontos de interrupção .....                            | 132 |
| Percorrendo o código .....                                           | 134 |
| Para os mais curiosos: main() e UIApplication .....                  | 136 |
| Desafio de prata: pinça para zoom .....                              | 136 |
| 8. UITableView e UITableViewController .....                         | 137 |
| Iniciando o aplicativo Homepwner .....                               | 137 |
| UITableViewController .....                                          | 138 |
| Criação de subclasses de UITableViewController .....                 | 139 |
| Fonte de dados da UITableView .....                                  | 141 |
| Criação de uma BNRIItemStore .....                                   | 142 |
| Implementação de métodos de fonte de dados .....                     | 145 |
| UITableViewCells .....                                               | 147 |
| Criação e recuperação de UITableViewCells .....                      | 148 |
| Reutilização de UITableViewCells .....                               | 150 |
| Biblioteca de fragmentos de código .....                             | 151 |
| Desafio de bronze: seções .....                                      | 154 |
| Desafio de prata: linhas constantes .....                            | 154 |
| Desafio de ouro: personalizar a tabela .....                         | 154 |
| 9. Edição de UITableView .....                                       | 155 |
| Modo de edição .....                                                 | 155 |
| Adição de linhas .....                                               | 161 |
| Exclusão de linhas .....                                             | 162 |
| Movimentação de linhas .....                                         | 163 |
| Desafio de bronze: renomear o botão Delete .....                     | 164 |
| Desafio de prata: impedir a reordenação .....                        | 165 |
| Desafio de ouro: realmente impedir a reordenação .....               | 165 |
| 10. UINavigationController .....                                     | 167 |
| UINavigationController .....                                         | 168 |
| UIViewController adicional .....                                     | 171 |
| Navegação com UINavigationController .....                           | 176 |
| Envio de controladores de visão .....                                | 176 |
| Passando dados entre controladores de visão .....                    | 177 |
| Fazendo com que visões apareçam e desapareçam .....                  | 178 |
| UINavigationBar .....                                                | 179 |
| Desafio de bronze: exibir um teclado numérico .....                  | 183 |
| Desafio de prata: fechar um teclado numérico .....                   | 183 |
| Desafio de ouro: enviar mais controladores de visão .....            | 183 |
| 11. Câmera .....                                                     | 185 |
| Exibição de imagens e UIImageView .....                              | 185 |
| Adição de um botão de câmera .....                                   | 187 |
| Captura de fotos e UIImagePickerController .....                     | 189 |
| Configuração de sourceType do seletor de imagens .....               | 189 |
| Configuração do delegate do seletor de imagens .....                 | 190 |
| Apresentação do seletor de imagens modalmente .....                  | 191 |
| Salvamento da imagem .....                                           | 192 |
| Criação de BNRIImageStore .....                                      | 193 |
| NSDictionary .....                                                   | 194 |
| Criação e uso de chaves .....                                        | 196 |
| Finalização de BNRIImageStore .....                                  | 199 |
| Dispensando o teclado .....                                          | 200 |
| Desafio de bronze: edição de imagens .....                           | 201 |
| Desafio de prata: remoção de imagens .....                           | 201 |
| Desafio de ouro: sobreposição de câmera .....                        | 201 |
| Para os mais curiosos: navegação por arquivos de implementação ..... | 201 |
| #pragma mark .....                                                   | 203 |

|                                                                                                  |     |
|--------------------------------------------------------------------------------------------------|-----|
| Para os mais curiosos: gravação de vídeo .....                                                   | 204 |
| 12. Eventos de toque e UIResponder .....                                                         | 207 |
| Eventos de toque .....                                                                           | 207 |
| Criação do aplicativo TouchTracker .....                                                         | 208 |
| Desenho com TouchDrawView .....                                                                  | 210 |
| Transformando toques em linhas .....                                                             | 211 |
| Lidando com toques múltiplos .....                                                               | 212 |
| Desafio de bronze: salvar e carregar .....                                                       | 215 |
| Desafio de prata: cores .....                                                                    | 215 |
| Desafio de ouro: círculos .....                                                                  | 216 |
| Para os mais curiosos: a cadeia de respondentes .....                                            | 216 |
| Para os mais curiosos: UIControl .....                                                           | 216 |
| 13. UIGestureRecognizer e UIMenuController .....                                                 | 219 |
| Subclasses de UIGestureRecognizer .....                                                          | 219 |
| Detecção de toques curtos com UITapGestureRecognizer .....                                       | 220 |
| Reconhecedores de múltiplos gestos .....                                                         | 221 |
| UIMenuController .....                                                                           | 223 |
| UILongPressGestureRecognizer .....                                                               | 224 |
| UIPanGestureRecognizer e reconhecedores simultâneos .....                                        | 225 |
| Para os mais curiosos: UIMenuController e UIResponderStandardEditActions .....                   | 228 |
| Para os mais curiosos: mais sobre UIGestureRecognizer .....                                      | 228 |
| Desafio de prata: linhas misteriosas .....                                                       | 229 |
| Desafio de ouro: velocidade e tamanho .....                                                      | 229 |
| Super desafio de ouro: cores .....                                                               | 229 |
| 14. Ferramentas de depuração .....                                                               | 231 |
| Medidores .....                                                                                  | 231 |
| Instrumentos .....                                                                               | 233 |
| Instrumento Allocations .....                                                                    | 233 |
| Instrumento Time Profiler .....                                                                  | 238 |
| Instrumento Leaks .....                                                                          | 241 |
| Analizador estático .....                                                                        | 242 |
| Projetos, destinos e ajustes de compilação .....                                                 | 243 |
| Ajustes de compilação .....                                                                      | 245 |
| Alteração de um ajuste de compilação .....                                                       | 246 |
| 15. Introdução ao Auto Layout .....                                                              | 249 |
| Universalização do Homepwner .....                                                               | 249 |
| O sistema de layout automático .....                                                             | 251 |
| Retângulo de alinhamento e atributos de layout .....                                             | 251 |
| Restrições .....                                                                                 | 252 |
| Adição de restrições no Interface Builder .....                                                  | 254 |
| Adição de mais restrições .....                                                                  | 257 |
| Adição de ainda mais restrições .....                                                            | 261 |
| Prioridades .....                                                                                | 262 |
| Depuração de restrições .....                                                                    | 262 |
| Layout ambíguo .....                                                                             | 263 |
| Restrições insatisfatórias .....                                                                 | 266 |
| Visões posicionadas incorretamente .....                                                         | 266 |
| Desafio de bronze: a prática leva à perfeição .....                                              | 268 |
| Desafio de prata: universalizar o Quiz .....                                                     | 268 |
| Para os mais curiosos: depurar com o Auto Layout Trace (rastreamento de layout automático) ..... | 269 |
| Para os mais curiosos: múltiplos arquivos XIB .....                                              | 269 |
| 16. Auto Layout: restrições programáticas .....                                                  | 271 |
| Visual Format Language .....                                                                     | 272 |
| Criação de restrições .....                                                                      | 273 |
| Adição de restrições .....                                                                       | 274 |
| Tamanho de conteúdo intrínseco .....                                                             | 276 |
| A outra maneira .....                                                                            | 277 |
| Para os mais curiosos: NSAutoresizingMaskLayoutConstraint .....                                  | 279 |

|                                                                                    |     |
|------------------------------------------------------------------------------------|-----|
| 17. Rotação automática, controladores popover e controladores de visão modal ..... | 281 |
| Rotação automática .....                                                           | 281 |
| Notificação de rotação .....                                                       | 283 |
| UIPopoverController .....                                                          | 285 |
| Mais controladores de visão modais .....                                           | 287 |
| Dispensando controladores de visão modais .....                                    | 290 |
| Estilos de controlador de visão modal .....                                        | 291 |
| Blocos de conclusão .....                                                          | 293 |
| Transições do controlador de visão modal .....                                     | 294 |
| Singletons seguros para threads .....                                              | 294 |
| Desafio de bronze: outro singleton seguro para thread .....                        | 295 |
| Desafio de ouro: aparência do popover .....                                        | 295 |
| Para os mais curiosos: bit masks (máscaras de bits) .....                          | 295 |
| Para os mais curiosos: relacionamentos de controlador de visão .....               | 297 |
| Relacionamentos pai-filho .....                                                    | 297 |
| Relacionamentos apresentado-apresentador .....                                     | 298 |
| Relacionamentos interfamiliares .....                                              | 298 |
| 18. Salvamento, carregamento e estados do aplicativo .....                         | 301 |
| Arquivamento .....                                                                 | 301 |
| Área restrita do aplicativo .....                                                  | 303 |
| Construção de um caminho de arquivo .....                                          | 304 |
| NSKeyedArchiver e NSKeyedUnarchiver .....                                          | 305 |
| Transições e estados do aplicativo .....                                           | 307 |
| Gravação no sistema de arquivos com NSData .....                                   | 309 |
| NSNotificationCenter e avisos de pouca memória .....                               | 311 |
| Mais informações sobre NSNotificationCenter .....                                  | 313 |
| Padrão de projeto Modelo-Visão-Controlador-Armazenamento .....                     | 314 |
| Desafio de bronze: PNG .....                                                       | 314 |
| Para os mais curiosos: transições de estado do aplicativo .....                    | 314 |
| Para os mais curiosos: leitura e gravação no sistema de arquivos .....             | 315 |
| Para os mais curiosos: o pacote de aplicativo .....                                | 317 |
| 19. Criação de subclasses UITableViewCell .....                                    | 319 |
| Criação de BNRItemCell .....                                                       | 319 |
| Configuração da interface de uma subclasse UITableViewCell .....                   | 320 |
| Exposição das propriedades de BNRItemCell .....                                    | 321 |
| Utilização de BNRItemCell .....                                                    | 322 |
| Restrições para BNRItemCell .....                                                  | 323 |
| Manipulação de imagens .....                                                       | 326 |
| Retransmissão de ações de UITableViewCells .....                                   | 328 |
| Adição de um bloco à subclasse da célula .....                                     | 329 |
| Apresentação de imagens em um controlador popover .....                            | 330 |
| Captura de variáveis .....                                                         | 332 |
| Desafio de bronze: código de cores .....                                           | 334 |
| Desafio de ouro: zoom .....                                                        | 334 |
| Para os mais curiosos: UICollectionView .....                                      | 334 |
| 20. Dynamic Type (tipo dinâmico) .....                                             | 337 |
| Utilização de fontes preferenciais .....                                           | 338 |
| Respondendo a alterações do usuário .....                                          | 340 |
| Atualização do Auto Layout .....                                                   | 340 |
| Revendo envolvimento de conteúdo e prioridades de resistência à compressão .....   | 340 |
| Determinação do tamanho do texto preferencial do usuário .....                     | 342 |
| Atualização de BNRItemCell .....                                                   | 343 |
| Outlets de restrição .....                                                         | 345 |
| Restrições de placeholder .....                                                    | 346 |
| 21. Serviços web e UIWebView .....                                                 | 349 |
| Serviços web .....                                                                 | 350 |
| Iniciando a construção do aplicativo Nerdfeed .....                                | 350 |
| NSURL, NSURLRequest, NSURLSession e NSURLSessionTask .....                         | 352 |

---

|                                                                                          |     |
|------------------------------------------------------------------------------------------|-----|
| Formatação de URLs e solicitações .....                                                  | 352 |
| Trabalhando com NSURLSession .....                                                       | 353 |
| Dados JSON .....                                                                         | 354 |
| Análise de dados JSON .....                                                              | 355 |
| O thread principal .....                                                                 | 357 |
| UIWebView .....                                                                          | 357 |
| Credenciais .....                                                                        | 359 |
| Desafio de prata: mais UIWebView .....                                                   | 361 |
| Desafio de ouro: próximos cursos .....                                                   | 361 |
| Para os mais curiosos: o corpo da solicitação .....                                      | 361 |
| 22. UISplitViewController .....                                                          | 363 |
| Divisão do Nerdfeed .....                                                                | 363 |
| Exibição do controlador de visão mestre no modo retrato .....                            | 366 |
| Universalização do Nerdfeed .....                                                        | 369 |
| 23. Core Data .....                                                                      | 371 |
| Mapeamento objeto-relacional .....                                                       | 371 |
| Mudança do Homepwner para o Core Data .....                                              | 371 |
| O arquivo do modelo .....                                                                | 371 |
| NSManagedObject e subclasses .....                                                       | 376 |
| Atualização de BNRItemStore .....                                                        | 378 |
| Adição de BNRAssetTypes ao Homepwner .....                                               | 382 |
| Mais sobre SQL .....                                                                     | 386 |
| Objetos falsos .....                                                                     | 387 |
| Vantagens e desvantagens de mecanismos persistentes .....                                | 389 |
| Desafio de bronze: ativos no iPad .....                                                  | 389 |
| Desafio de prata: tipos de ativos novos .....                                            | 389 |
| Desafio de ouro: exibição de ativos de um tipo .....                                     | 389 |
| 24. Restauração de estado .....                                                          | 391 |
| Como a restauração de estado funciona .....                                              | 391 |
| Optando pela restauração de estado .....                                                 | 392 |
| Identificadores e classes de restauração .....                                           | 393 |
| Ciclo de vida da restauração de estado .....                                             | 395 |
| Restauração de controladores de visão .....                                              | 396 |
| Codificação de dados relevantes .....                                                    | 398 |
| Salvamento de estados de visão .....                                                     | 399 |
| Desafio de prata: outro aplicativo .....                                                 | 401 |
| Para os mais curiosos: controle de instantâneos .....                                    | 401 |
| 25. Localização .....                                                                    | 403 |
| Internacionalização usando NSNumberFormat .....                                          | 403 |
| Recursos de localização .....                                                            | 406 |
| NSBundle e tabelas de strings .....                                                      | 409 |
| Desafio de bronze: outra localização .....                                               | 411 |
| Para os mais curiosos: o papel de NSBundle na internacionalização .....                  | 412 |
| Para os mais curiosos: localização de arquivos XIB sem o Base Internationalization ..... | 412 |
| 26. UserDefaults .....                                                                   | 415 |
| NSUserDefaults .....                                                                     | 415 |
| Registro das configurações de fábrica .....                                              | 416 |
| Leitura de uma preferência .....                                                         | 416 |
| Alteração de uma preferência .....                                                       | 417 |
| Pacote de configurações .....                                                            | 417 |
| Edição da Root.plist .....                                                               | 419 |
| Root.strings localizadas .....                                                           | 420 |
| 27. Controle de animações .....                                                          | 421 |
| Animações básicas .....                                                                  | 421 |
| Funções de tempo .....                                                                   | 422 |
| Animação de quadro-chave .....                                                           | 423 |
| Conclusão da animação .....                                                              | 424 |
| Animações de mola .....                                                                  | 425 |

|                                                    |     |
|----------------------------------------------------|-----|
| Desafio de prata: Quiz aprimorado .....            | 427 |
| 28. UIStoryboard .....                             | 429 |
| Criação de um storyboard .....                     | 429 |
| UITableViewController em storyboards .....         | 432 |
| Segues .....                                       | 435 |
| Permitindo mudanças de cor .....                   | 440 |
| Repassando dados .....                             | 442 |
| Mais sobre storyboards .....                       | 447 |
| Para os mais curiosos: restauração de estado ..... | 447 |
| 29. Posfácio .....                                 | 449 |
| Quais são os próximos passos .....                 | 449 |
| A minha propaganda .....                           | 449 |
| Index .....                                        | 451 |



# Introdução

Como aspirante a desenvolvedor de iOS, você enfrenta três dificuldades básicas:

- *Deve aprender a linguagem Objective-C.* Objective-C é uma extensão pequena e simples da linguagem de programação C. Depois dos quatro primeiros capítulos deste livro, você terá um conhecimento prático de Objective-C.
- *Deve dominar grandes ideias.* Essas ideias incluem coisas como técnicas de gerenciamento de memória, delegação, arquivamento e o uso adequado de controladores de visão. As grandes ideias demoram alguns dias para serem entendidas. Quando chegar à metade deste livro, você entenderá essas grandes ideias.
- *Deve dominar os frameworks.* O objetivo final é saber como usar todos os métodos, de todas as classes, em todos os frameworks no iOS. Este é um projeto para a vida toda: há mais de 3.000 métodos e mais de 200 classes disponíveis no iOS. Para complicar mais ainda, a Apple adiciona novas classes e novos métodos em todas as versões de iOS. Neste livro, você será apresentado a cada um dos subsistemas que formam o kit de desenvolvimento de software (SDK) do iOS, mas não os estudaremos em profundidade. Em vez disso, nosso objetivo é fazer com que você chegue ao ponto onde possa pesquisar e entender a documentação de referência da Apple.

Usamos este material diversas vezes em nosso campo de treinamento para desenvolvimento de iOS, na Big Nerd Ranch. Ele foi bastante testado e já ajudou centenas de pessoas a se tornarem desenvolvedores de aplicativos para iOS. Esperamos, sinceramente, que se prove útil para você.

## Pré-requisitos

Este livro supõe que você já esteja motivado a aprender a escrever aplicativos para iOS. Não perderemos tempo convencendo-o de que iPhone, iPad e iPod Touch são itens de tecnologia fascinantes.

Supomos também que você conhece a linguagem de programação C e alguma coisa sobre programação orientada a objeto. Se isso não for verdade, você provavelmente deve começar com um livro de introdução das linguagens C e Objective-C, como o *Objective-C Programming: The Big Nerd Ranch Guide* (Programação em Objective-C: Guia da Big Nerd Ranch).

## O que mudou na quarta edição?

Esta edição considera que o leitor está usando Xcode 5 e executando aplicativos em um dispositivo ou simulador de iOS 7.

Adotamos um estilo mais moderno de Objective-C nesta edição. Nós usamos amplamente propriedades, notação de ponto, variáveis de instância com sintetização automática, os novos literais e subíndice. Também usamos mais blocos.

A Apple continua evoluindo o iOS, e nós adicionamos com entusiasmo a abordagem de animações baseadas em blocos, Auto Layout (layout automático) e **NSURLSession** ao livro.

Além dessas alterações óbvias, fizemos milhares de pequenas melhorias, que foram inspiradas por perguntas de nossos leitores ou alunos. Cada capítulo deste livro está simplesmente um pouco melhor que o capítulo correspondente da terceira edição.

## Nossa filosofia de ensino

Este livro lhe ensinará noções básicas de programação de iOS. Ao mesmo tempo, você digitará muitos códigos e construirá diversos aplicativos. Ao final do livro, você terá conhecimento e experiência. Contudo, todo conhecimento não deve vir em primeiro lugar (e neste livro não virá). Essa é uma espécie de estilo tradicional que passamos a conhecer e odiar. Em vez disso, adotamos a abordagem de aprender fazendo. Conceitos de desenvolvimento e codificação real andam juntos.

Aqui se encontra o que aprendemos ao longo dos anos de ensino de programação de iOS:

- Aprendemos que ideias as pessoas devem ter para começar a programar e nos focamos neste subconjunto.

- Descobrimos que as pessoas aprendem melhor quando tais conceitos são apresentados *conforme são necessários*.
- Aprendemos que o conhecimento e a experiência em programação se desenvolvem melhor quando crescem juntos.
- Aprendemos que “entrar na rotina” é muito mais importante do que parece. Muitas vezes, pediremos que digite um código antes de entendê-lo. Percebemos que você pode vir a se sentir um macaco amestrado, digitando um monte de códigos que não comprehende totalmente. Mas a melhor forma de aprender codificação é encontrando e corrigindo erros de digitação. Longe de ser maçante, esta depuração básica é onde você aprende realmente os macetes do código. É por isso que incentivamos você a digitar o código. Você poderia simplesmente baixá-lo, mas copiar e colar não é programar. Queremos mais para você e suas habilidades.

O que isso significa para você, leitor? Para aprender assim, é preciso confiança. E agradecemos a sua. Também é preciso paciência. Ao guiá-lo por estes capítulos, tentaremos mantê-lo à vontade e informá-lo sobre o que está acontecendo. No entanto, haverá momentos em que você terá de confiar em nós. (Se acha que isso vai incomodá-lo, continue lendo – temos algumas ideias que podem ajudá-lo.) Não desanime se encontrar um conceito que você não entende imediatamente. Lembre-se de que, intencionalmente, *não* estamos proporcionando de uma vez só todo o conhecimento que você precisará. Se um conceito parecer confuso, provavelmente vamos discuti-lo detalhadamente mais tarde, quando ele for necessário. E alguns pontos que não ficarem claros no começo, de repente farão sentido, quando você implementá-los pela primeira vez (ou décima-segunda vez).

As pessoas aprendem de forma diferente. É possível que você adore a forma como apresentamos conceitos à medida que forem necessários. Também é possível que você ache isso frustrante. Se for este o caso, veja algumas opções:

- Respire fundo e espere. Chegaremos lá, e você também.
- Verifique o índice. Deixaremos passar se você se adiantar e ler uma discussão mais avançada, que acontece mais adiante no livro.
- Verifique a documentação on-line da Apple. Essa é uma ferramenta essencial do desenvolvedor e você vai querer muita prática de uso. Consulte-a antecipadamente e frequentemente.
- Se são conceitos de Objective-C ou de programação orientada a objetos que estão dificultando as coisas para você (ou se acha que vão dificultar), você pode pensar em fazer backup ou ler o *Objective-C Programming: The Big Nerd Ranch Guide* (Programação em Objective-C: Guia da Big Nerd Ranch).

## Como usar este livro

Este livro é baseado na aula que ministramos na Big Nerd Ranch. Sendo assim, ele foi desenvolvido para ser usado de uma determinada maneira.

Estabeleça uma meta razoável para você, como: “Farei um capítulo por dia”. Quando sentar para atacar um capítulo, procure um lugar tranquilo, onde você não será interrompido por pelo menos uma hora. Feche seu e-mail, o cliente de Twitter e o programa de bate-papo. Não é hora de realizar várias tarefas ao mesmo tempo, você precisará se concentrar.

Faça a programação real. Você pode ler um capítulo todo primeiro, se quiser. Mas a aprendizagem mesmo começa quando você senta e faz a codificação. Você não entenderá o conceito de verdade até que tenha escrito um programa que o utiliza e, talvez mais importante, tenha depurado este programa.

Alguns exercícios exigem arquivos de suporte. Por exemplo, no primeiro capítulo, você precisará de um ícone para o aplicativo Quiz (testes) e temos um para você. Você pode baixar os recursos e soluções dos exercícios em <http://www.bignerdranch.com/solutions/iOSProgramming4ed.zip>.

Há dois tipos de aprendizagem. Quando você aprende sobre a Guerra do Peloponeso, está simplesmente acrescentando detalhes a uma estrutura de ideias que já entende. Isso é o que chamaremos de “fácil aprendizagem”. Sim, aprender sobre a Guerra do Peloponeso pode demorar bastante, mas você raramente fica

confuso com ela. Aprender a programação de iOS, por outro lado, é de “difícil aprendizagem” e você pode ficar bastante confuso às vezes, principalmente nos primeiros dias. Ao escrever este livro, tentamos criar uma experiência que atenuará os altos e baixos da curva de aprendizagem. Aqui estão duas coisas que você pode fazer para tornar a jornada mais fácil:

- Encontre alguém que já saiba escrever aplicativos de iOS e que responderá às suas perguntas. Em especial, colocar o aplicativo no dispositivo pela primeira vez é, normalmente, muito frustrante se você estiver fazendo isso sem a ajuda de um desenvolvedor experiente.
- Durma bem. Pessoas sonolentas não lembram o que aprenderam.

## Como este livro está organizado

Neste livro, cada capítulo aborda uma ou mais ideias de desenvolvimento de iOS, através de discussões e exercícios práticos. Para treinar mais a escrita de códigos, a maioria dos capítulos conta com exercícios desafiadores. Nós o incentivamos a resolver pelo menos alguns deles. São excelentes para fixar os conceitos apresentados no capítulo e torná-lo um programador de iOS mais confiante. Finalmente, a maioria dos capítulos termina com uma ou duas seções “Para os mais curiosos”, que explicam certas consequências dos conceitos que foram apresentados anteriormente.

O Capítulo 1 apresenta a programação de iOS conforme você constrói e implementa um pequeno aplicativo. Você terá o primeiro contato com o **Xcode** e o simulador de iOS, juntamente com todas as etapas de criação de projetos e arquivos. O capítulo inclui uma discussão de **MVC** (Modelo-Visão-Controlador) e sua relação com o desenvolvimento de iOS.

Os Capítulos 2 e 3 oferecem uma visão geral do Objective-C e o gerenciamento de memória. Embora você não crie um aplicativo de iOS nesses dois capítulos, você fará a construção e a depuração de uma ferramenta chamada **RandomItems** para lhe dar os fundamentos desses conceitos.

Nos Capítulos 4 e 5, você começará a se concentrar na interface do usuário do iOS, à medida que aprende sobre visões e hierarquia de visões e cria um aplicativo chamado **Hypnosister**.

Os Capítulos 6 e 7 apresentam os controladores de visão para o gerenciamento de interfaces de usuário por meio do aplicativo **HypnoNerd**. Você ganhará prática trabalhando com visões e controladores de visões, e também com a navegação entre telas usando uma barra de guias. Você também obterá muita experiência com o importante padrão de projeto de delegação, assim como trabalhará com protocolos, o depurador e a configuração de notificações locais.

O Capítulo 8 apresenta o maior aplicativo do livro, o **Homepwner**. (A propósito, “Homepwner” não é um erro de digitação; você encontra a definição de “pwn” em [www.urbandictionary.com](http://www.urbandictionary.com).) Este aplicativo mantém um registro de seus bens em caso de incêndio ou outra catástrofe. O **Homepwner** levará quatorze capítulos para terminar.

Nos Capítulos 8, 9 e 19, você adquirirá experiência com tabelas. Aprenderá sobre visões de tabela, seus controladores de visão e fontes de dados. Aprenderá como exibir dados em uma tabela, como permitir que o usuário edite a tabela e como melhorar a interface.

O Capítulo 10 baseia-se na experiência em navegação, adquirida no Capítulo 6. Você aprenderá como usar a **UINavigationController** e dará ao **Homepwner** uma interface detalhada e uma barra de navegação.

No Capítulo 11, você aprenderá como tirar fotos com a câmera e como exibir e armazenar imagens no **Homepwner**. Você usará **NSDictionary** e **UIImagePickerController**.

Nos Capítulos 12 e 13, você deixará o **Homepwner** um pouco de lado para criar um aplicativo de desenho chamado **TouchTracker** para aprender sobre eventos de toque. Você verá como adicionar recursos de multitoque e como usar a **UIGestureRecognizer** para responder a gestos específicos. Você também adquirirá experiência com os conceitos de primeiro respondente e cadeia de respondentes, além de mais prática com a **NSDictionary**.

No Capítulo 14, você aprenderá como usar os medidores de depuração, o **Instruments** e o analisador estático para otimizar o desempenho do **TouchTracker**.

Nos Capítulos 15 e 16, você transformará o **Homepwner** em um aplicativo universal – um aplicativo que executa nativamente tanto em iPhone quanto em iPad. Você também trabalhará com o Auto Layout para compilar uma interface que será exibida corretamente em qualquer tamanho de tela.

No Capítulo 17, você aprenderá sobre como lidar com rotação e usar a **UIPopoverController** para o iPad e controladores de visão modais.

O Capítulo 18 examina maneiras de salvar e carregar dados. Particularmente, você arquivará dados no aplicativo Homepwner.

No Capítulo 20, você atualizará o Homepwner para que ele use o Dynamic Type (Tipo Dinâmico) para suportar diferentes tamanhos de fonte que o usuário preferir.

O Capítulo 21 deixa de tratar novamente do Homepwner e apresenta os serviços web, à medida que você cria o Nerdfeed. Esse aplicativo busca e analisa um feed RSS a partir de um servidor usando **NSURLConnection** e **NSXMLParser**. Nerdfeed também exibirá uma página da web em um **UIWebView**.

No Capítulo 22, você aprenderá sobre **UISplitViewController** e adicionará uma interface do usuário com visão dividida para que Nerdfeed tire proveito da tela grande do iPad.

O Capítulo 23 retorna para o aplicativo Homepwner com a apresentação do Core Data. Você mudará o Homepwner para armazenar e carregar dados usando uma **NSManagedObjectContext**.

No Capítulo 24, você adicionará a restauração de estado ao Homepwner para permitir que os usuários retornem ao aplicativo exatamente onde eles o deixaram, não importando quanto tempo ficaram longe dele.

O Capítulo 25 apresenta os conceitos e as técnicas de internacionalização e localização. Você aprenderá sobre **NSLocale**, tabelas de strings e **NSBundle** ao fazer a localização de partes do Homepwner.

No Capítulo 26, você usará **NSUserDefaults** para salvar suas preferências de forma persistente. Este capítulo concluirá o aplicativo Homepwner.

O Capítulo 27 apresenta o framework Core Animation, com um breve retorno ao aplicativo HypnoNerd para a implementação de animações.

O Capítulo 28 apresenta aplicativos de compilação por meio de storyboards. Você montará um aplicativo usando a **UIStoryboard** e saberá mais sobre os prós e contras do uso de storyboards.

## Opções de estilo

Este livro contém muitos códigos. Tentamos tornar este código e os designs por trás dele exemplares. Fizemos o melhor que pudemos para seguir a linguagem da comunidade, mas às vezes fugimos daquilo que pode ser visto no código de amostra da Apple ou em códigos encontrados em outros livros. Você pode não entender estes pontos agora, mas é melhor falarmos sobre eles antes que você comece a ler o livro:

- Geralmente criamos controladores de visão de forma programática. Alguns programadores criam instâncias de controladores de visão dentro de arquivos XIB ou em um storyboard. Iremos discutir storyboards e demonstrar um pouco seu uso, mas, na realidade, raramente os utilizamos em nossos projetos da Big Nerd Ranch.
- Quase sempre iniciaremos um projeto com o template de projeto mais simples: o aplicativo vazio. Quando seu aplicativo funcionar, você saberá que isso aconteceu por causa de seus esforços, não porque esse comportamento foi embutido no template.

## Convenções tipográficas

Para tornar este livro de fácil leitura, determinados itens aparecem em determinadas fontes. Nomes de classes, nomes de métodos e nomes de funções aparecem em fonte de largura fixa e em negrito. Nomes de classes começam com letra maiúscula e nomes de métodos com letra minúscula. Neste livro, nomes de métodos e funções têm a mesma formatação por questões de simplicidade. Por exemplo: “No método **loadView** da classe **BNRReViewController**, use a função **NSLog** para gravar o valor no console.”

Variáveis, constantes e tipos aparecem em fonte de largura fixa, mas não em negrito. Assim, você verá: “A variável **fido** será do tipo **float**. Inicialize-a para **M\_PI**.”

Aplicativos e opções de menu aparecem na fonte do sistema Mac. Por exemplo: “Abra Xcode e selecione New Project... a partir do menu File.”

Todos os blocos de código estarão em fonte de largura fixa. O código que você precisa digitar sempre está em negrito. Por exemplo, no código a seguir, você digitaria tudo, exceto a primeira e a última linha. (Estas linhas já estão no código e aparecem aqui para que você saiba onde adicionar o que é novo.)

```
@interface BNRQuizViewController ()  
  
@property (nonatomic, weak) IBOutlet UILabel *questionLabel;  
@property (nonatomic, weak) IBOutlet UILabel *answerLabel;  
  
@end
```

## Hardware e software necessários

Você só pode desenvolver aplicativos de iOS em Mac com processadores Intel. Você precisará baixar o SDK do iOS da Apple, que inclui o Xcode (Ambiente de desenvolvimento integrado da Apple), o simulador de iOS e outras ferramentas de desenvolvimento.

Você deveria associar-se ao programa de desenvolvedores iOS da Apple, que custa US\$ 99/ano, por três motivos:

- Baixar as ferramentas de desenvolvedores mais recentes é gratuito para membros.
- Apenas aplicativos assinados são executados em um dispositivo e apenas membros podem assinar aplicativos. Se quiser testar o aplicativo em seu dispositivo, você precisará se associar.
- Você não pode colocar um aplicativo na loja antes de ser um membro.

Se você vai dedicar o seu tempo para estudar este livro inteiro, a associação no programa de desenvolvedores iOS, sem dúvida, vale o investimento. Acesse <http://developer.apple.com/programs/ios/> para se cadastrar.

E quanto aos dispositivos iOS? A maioria dos aplicativos que você desenvolverá na primeira metade do livro é para iPhone, mas você poderá executá-los no iPad. Na tela do iPad, os aplicativos para iPhone aparecem em uma janela do tamanho do iPhone. Não é um uso interessante do iPad, mas é bom para começar com o iOS. Nestes primeiros capítulos, você se focará em aprender os fundamentos do SDK do iOS, que são os mesmos para qualquer um dos dispositivos iOS. Mais adiante no livro, veremos algumas opções apenas para iPad e como fazer com que os aplicativos sejam executados de forma nativa em ambas as famílias de dispositivos iOS.

Já está animado? Bom. Vamos começar.

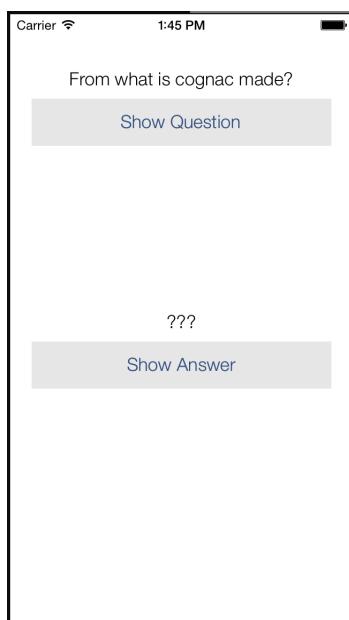


# 1

## Um aplicativo iOS simples

Neste capítulo, você escreverá um aplicativo iOS chamado Quiz. Esse aplicativo mostrará ao usuário uma pergunta e depois revelará a resposta quando o usuário pressionar um botão. Ao pressionar outro botão, a próxima pergunta será exibida (Figure 1.1).

Figure 1.1 Seu primeiro aplicativo: Quiz



Quando estiver escrevendo um aplicativo para iOS, você deve responder a duas perguntas básicas:

- Como eu posso criar e configurar corretamente os meus objetos? (Exemplo: “Eu quero aqui um botão chamado Show Question.”)
- Como lido com a interação do usuário? (Exemplo: “Quando o usuário tocar no botão, eu quero que esse pedaço de código seja executado.”)

A maior parte deste livro dedica-se a responder a essas perguntas.

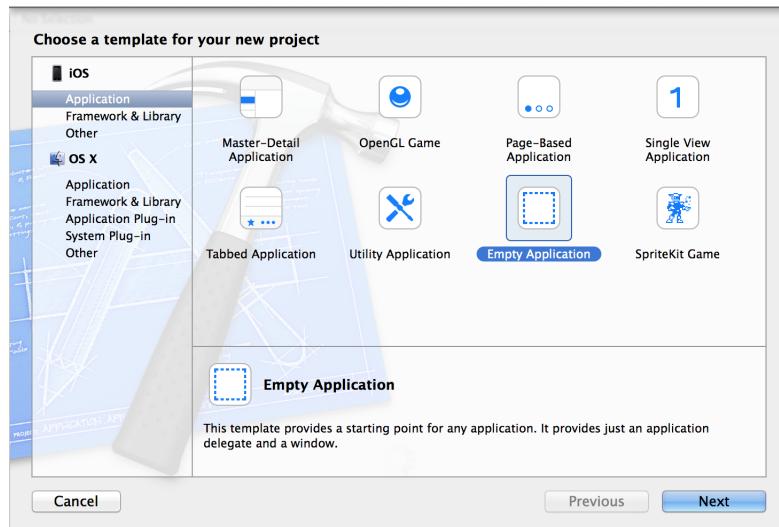
Conforme você percorrer este primeiro capítulo, provavelmente não entenderá tudo o que está fazendo, e pode se sentir ridículo ao ser apenas levado pela corrente. Mas ser levado pela corrente é o suficiente neste momento. A imitação é uma forma poderosa de aprendizagem; foi como você aprendeu a falar, e será como você vai começar a programar para iOS. Conforme for aprendendo, você pode experimentar e se desafiar a realizar coisas mais criativas na plataforma. Mas por enquanto, faça apenas o que estamos mostrando. Esses detalhes serão explicados em outros capítulos.

### Criação de um projeto no Xcode

Abra o Xcode e, no menu File, selecione New → Project...

Uma nova janela da área de trabalho aparecerá e uma página deslizará da barra de ferramentas. No lado esquerdo, encontre a seção iOS e selecione Application (Figure 1.2). Você terá vários templates de aplicativo para escolher. Selecione Empty Application.

Figure 1.2 Criação de um projeto



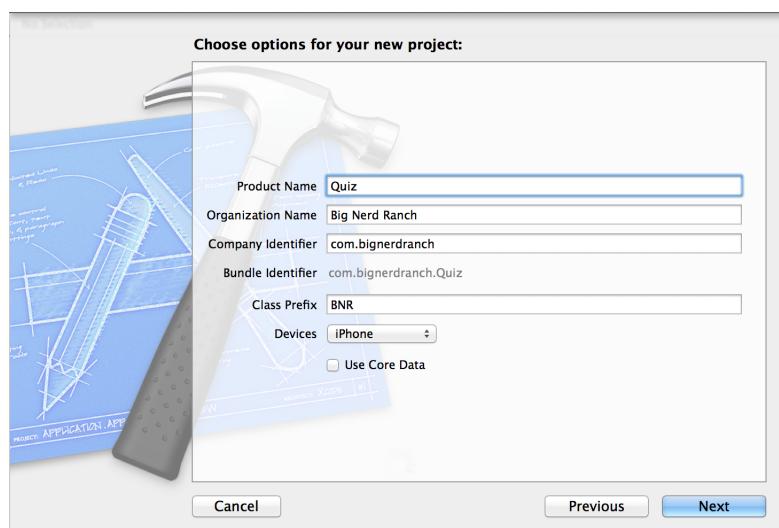
Você está utilizando o template Empty Application porque ele gera a menor quantidade possível de código padronizado. Muito código padronizado atrapalha o aprendizado do funcionamento das coisas.

Este livro foi criado para Xcode 5.0.2. Os nomes dos templates podem mudar com novas versões do Xcode. Se você não encontrar um template Empty Application, utilize o template que parecer mais simples. Ou visite o fórum da Big Nerd Ranch sobre este livro em [forums.bignerdranch.com](http://forums.bignerdranch.com) para obter ajuda com novas versões do Xcode.

Clique em Next e, na próxima página, entre no Quiz por Product Name (Figure 1.3). O nome da organização e o identificador da empresa são necessários para continuar. Você pode usar Big Nerd Ranch e com.bignerdranch. Ou utilize o nome de sua empresa e com.nomedesuaempresaqui.

No campo Class Prefix, digite BNR e, no menu pop-up rotulado Devices, escolha iPhone. Certifique-se de que a caixa de seleção Use Core Data não esteja marcada.

Figure 1.3 Configuração de um novo projeto



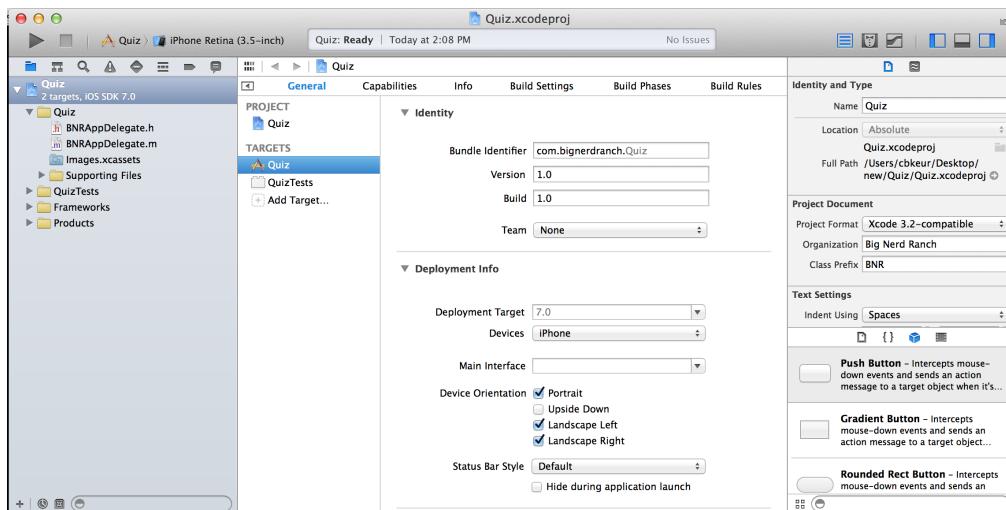
Você está criando o Quiz como um aplicativo de iPhone, mas ele será executado em um iPad. Não ficará perfeito na tela do iPad, mas por enquanto, isso não é problema. Na primeira parte deste livro, você se manterá

nos aplicativos para iPhone e seu foco será aprender os fundamentos do SDK do iOS, que são os mesmos para qualquer um dos dispositivos iOS. Mais adiante, você verá algumas opções de aplicativos exclusivos para iPad e também como rodar aplicativos nativamente em ambos os tipos de dispositivos.

Clique em **Next** e, na última página, salve o projeto no diretório onde você pretende guardar os exercícios deste livro. Você pode desmarcar a caixa que cria um repositório git local, mas mantê-la marcada não terá problema algum. Clique em **Create** para criar o projeto Quiz.

Depois que um projeto for criado, ele será aberto na janela da área de trabalho do Xcode (Figure 1.4).

Figure 1.4 Janela da área de trabalho do Xcode

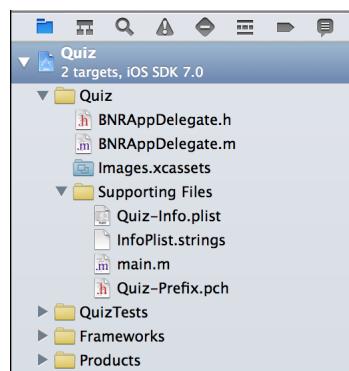


Observe o lado esquerdo da janela da área de trabalho. Essa área é conhecida como *área de navegação* e exibe vários *navegadores* diferentes – ferramentas que mostram as diferentes partes do seu projeto. Você pode escolher qual navegador usar selecionando um dos ícones do seletor de navegadores, que é a barra logo acima da área de navegação.

O navegador que está aberto é o *navegador de projetos*. O navegador de projetos mostra os arquivos que compõem o seu projeto (Figure 1.5). Você pode selecionar um arquivo para abri-lo na *área do editor* à direita da área de navegação.

Os arquivos no navegador de projetos podem ser agrupados em pastas para ajudá-lo a organizar seu projeto. Alguns grupos foram criados para você pelo template; você pode renomeá-los como quiser, ou acrescentar outros novos. Os grupos servem puramente para organizar os arquivos, e não se correlacionam com o sistema de arquivos de modo algum.

Figure 1.5 Arquivos do aplicativo Quiz no navegador de projetos



No navegador de projetos, encontre os arquivos chamados `BNRAppDelegate.h` e `BNRAppDelegate.m`. São os arquivos de uma *classe* chamada **BNRAppDelegate**. O template Empty Application criou essa classe para você.

Uma classe descreve um tipo de *objeto*. O desenvolvimento em iOS é orientado a objeto, e um aplicativo para iOS consiste, acima de tudo, em um conjunto de *objetos* trabalhando juntos. Quando o aplicativo Quiz for iniciado, um objeto do tipo **BNRAppDelegate** será criado. Nos referimos a um objeto **BNRAppDelegate** como uma *instância* da classe **BNRAppDelegate**.

Você aprenderá muito mais sobre como as classes e objetos funcionam no Chapter 2. No momento, você vai prosseguir para um pouco de teoria de projeto de aplicativos e então mergulhará no desenvolvimento.

## Modelo-Visão-Controlador

Modelo-Visão-Controlador, ou MVC, é um padrão de projeto utilizado em desenvolvimento de aplicativos iOS. Em MVC, cada objeto é um objeto modelo, um objeto de visão ou um objeto controlador.

- Os *objetos de visão* são visíveis para o usuário. Alguns exemplos de objetos de visão são: botões, campos de texto e controles deslizantes. Os objetos de visão compõem a interface do usuário de um aplicativo. No Quiz, os rótulos que mostram a pergunta e a resposta, assim como os botões abaixo deles, são objetos de visão.
- *Objetos de modelo* armazenam dados e não têm qualquer conhecimento sobre a interface do usuário. No Quiz, os objetos de modelo serão duas listas de strings: uma para perguntas e outra para respostas.

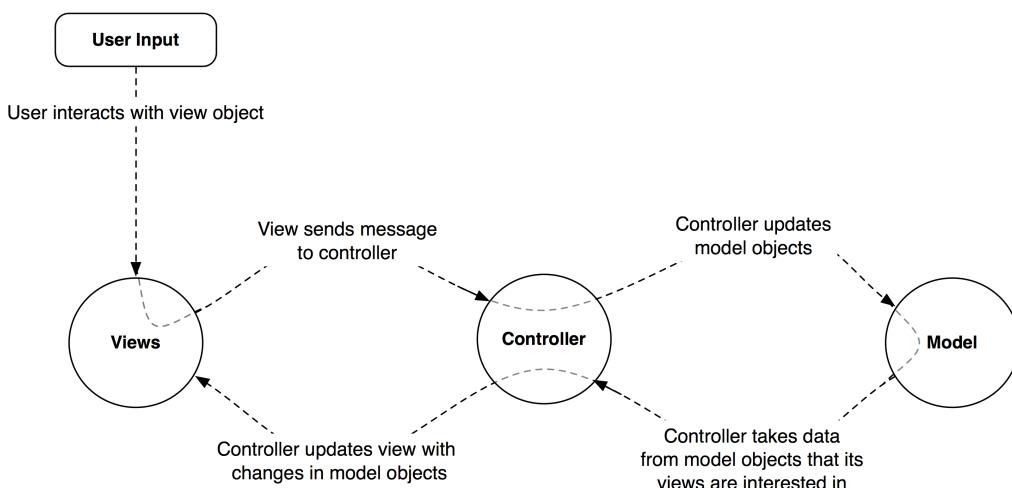
Geralmente, os objetos de modelo modelam objetos reais do mundo do usuário. Por exemplo, quando você escreve um aplicativo para uma empresa de seguros, você provavelmente acabará criando uma classe de modelo customizada chamada **InsurancePolicy**.

- *Objetos de controlador* são os gerentes de um aplicativo. Eles configuram as visões que o usuário enxerga e garantem a sincronização dos objetos de visão e de modelo.

Em geral, os controladores lidam com questões do tipo “E depois?”. Por exemplo, quando o usuário seleciona um item de uma lista, o controlador determina o que o usuário verá a seguir.

A Figure 1.6 mostra o fluxo de controle em um aplicativo em resposta a uma entrada do usuário, como o toque em um botão.

Figure 1.6 Padrão MVC



Observe que os objetos modelo e visão não se comunicam diretamente uns com os outros; os objetos controladores ficam no centro de tudo, recebendo as mensagens de alguns objetos e encaminhando instruções a outros.

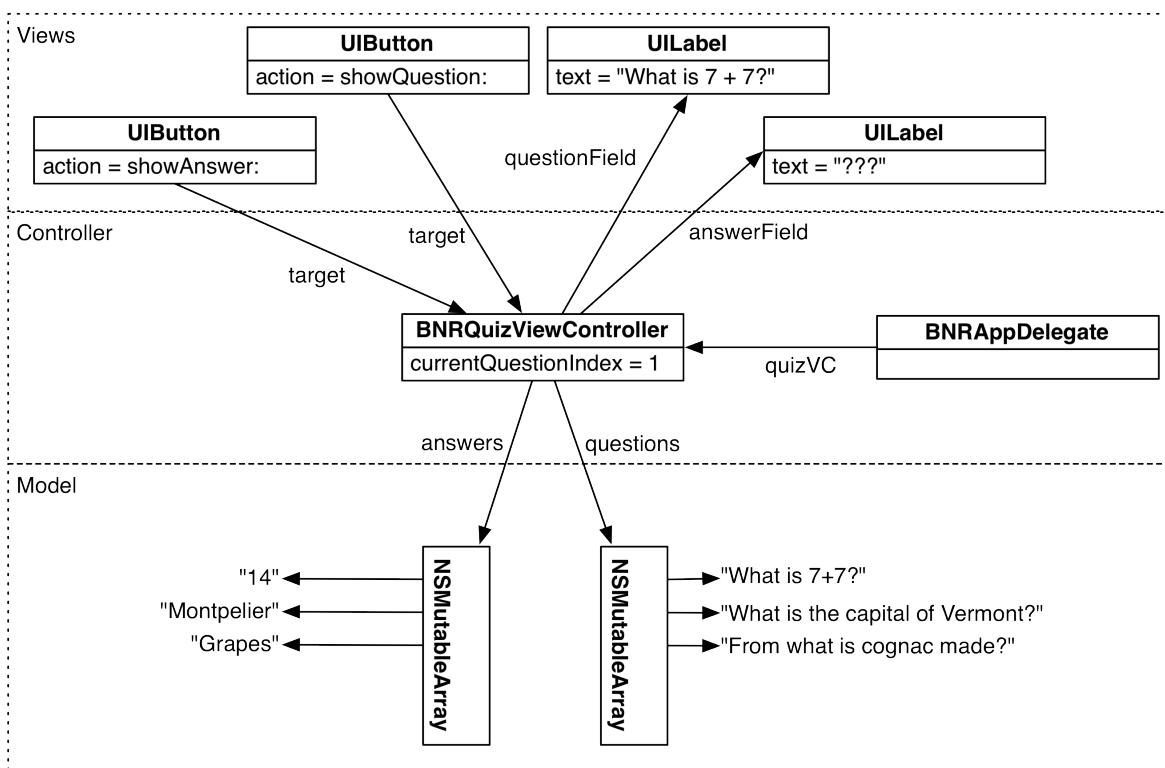
## Projetando o Quiz

Você escreverá o aplicativo Quiz utilizando o padrão MVC. Estes são os objetos que você criará e com os quais você trabalhará:

- 4 objetos de visão: duas instâncias de **UILabel** e duas instâncias de **UIButton**
- 2 objetos controladores: uma instância de **BNRAppDelegate** e uma instância de **BNRQuizViewController**
- 2 objetos de modelo: duas instâncias de **NSMutableArray**

Esse objetos e seus relacionamentos estão organizados no diagrama de objetos do Quiz mostrado na Figure 1.7.

Figure 1.7 Diagrama de objetos do Quiz



Esse diagrama é um panorama de como o aplicativo Quiz funcionará depois de pronto. Por exemplo, quando o botão Show Question for pressionado, ativará um *método* no **BNRQuizViewController**. Um método é bem parecido com uma função: uma lista de instruções que devem ser executadas. Esse método busca uma nova pergunta em um array de perguntas e pede ao rótulo superior para exibir a pergunta.

Não tem problema se o diagrama de objetos não fizer sentido por enquanto; ele fará sentido até o final do capítulo. Retome-o conforme você for vendo como o aplicativo está tomando forma.

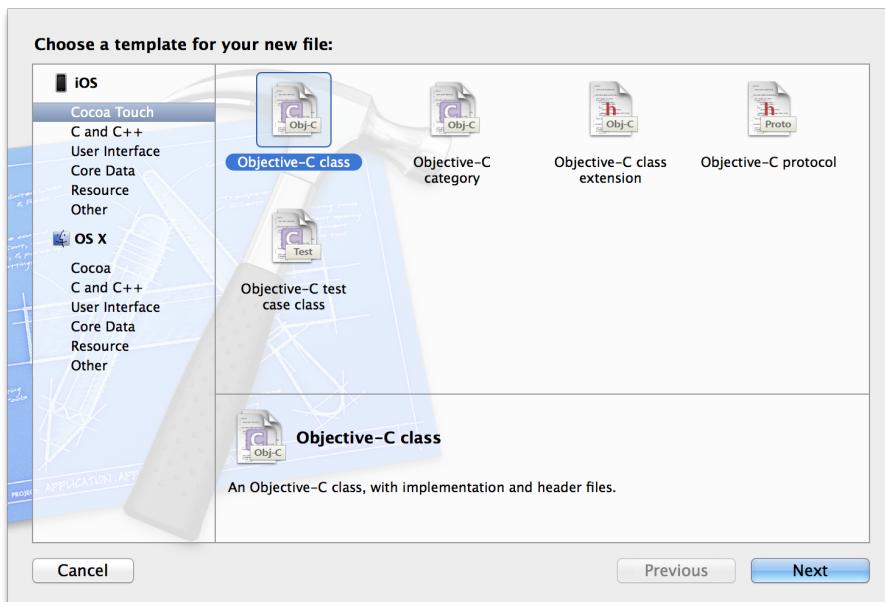
Você construirá o Quiz passo a passo, começando com o objeto controlador que fica no meio do aplicativo – **BNRQuizViewController**.

## Criação de um controlador de visão

A classe **BNRAppDelegate** foi criada para você pelo template Empty Application, mas você terá que criar a classe **BNRQuizViewController**. Falaremos mais sobre classes no Chapter 2 e mais sobre controladores de visão no Chapter 6. Por enquanto, basta acompanhar os passos.

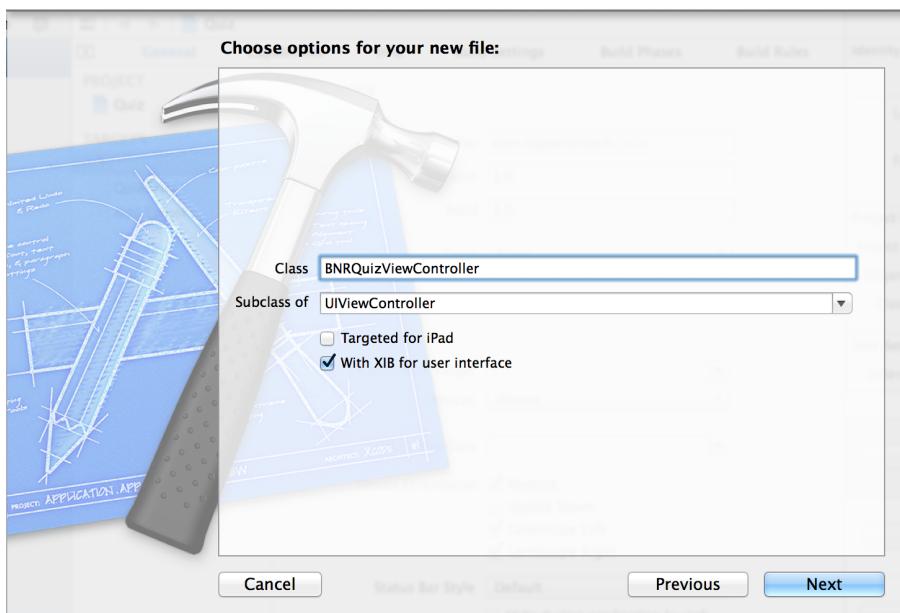
No menu File, selecione New → File.... Uma página deslizará perguntando que tipo de arquivo você gostaria de criar. No lado esquerdo, na seção iOS, selecione Cocoa Touch. Escolha Objective-C Class e clique em Next.

Figure 1.8 Criação de uma classe do Objective-C



Na próxima página, nomeie a classe como **BNRQuizViewController**. No campo Subclass of, clique na seta do menu suspenso e selecione **UIViewController**. Marque a caixa de seleção With XIB for user interface.

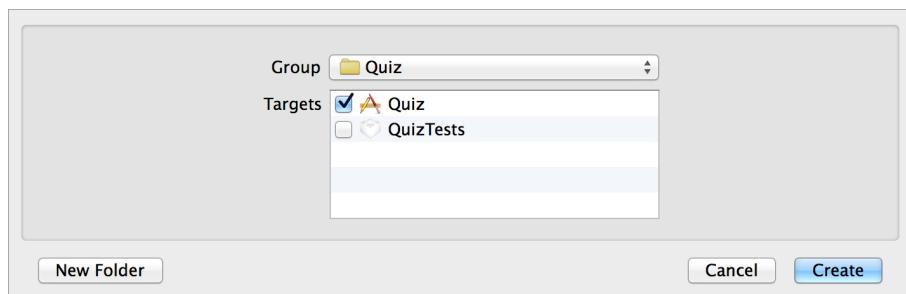
Figure 1.9 Criação de um controlador de visão



Clique em Next e será exibido um painel suspenso que solicitará que você crie os arquivos para essa nova classe. Ao criar uma nova classe para um projeto, você deve salvar os arquivos que a descrevem dentro do diretório-fonte do projeto no sistema de arquivos. Como padrão, o diretório do projeto atual já estará selecionado para você. Você também pode escolher o grupo no navegador de projetos aos quais esses arquivos serão adicionados. Como esses grupos são simplesmente para fins de organização e esse projeto é muito pequeno, usaremos apenas as definições padrão.

Certifique-se de que a caixa do destino Quiz esteja selecionada. Isso garante que a classe **BNRQuizViewController** seja compilada quando o projeto Quiz for construído. Clique em Create.

Figure 1.10 O destino Quiz está selecionado



## Construção de uma interface

No navegador de projetos, encontre os arquivos da classe **BNRQuizViewController**. Quando você criou essa classe, marcou a caixa With XIB for user interface, portanto, **BNRQuizViewController** veio com um terceiro arquivo de classe: **BNRQuizViewController.xib**. Encontre e selecione **BNRQuizViewController.xib** no navegador de projetos para abri-lo na área do editor.

Quando o Xcode abre um arquivo XIB (pronuncia-se “zib”), abre-o com o Interface Builder, uma ferramenta visual onde você pode adicionar e posicionar objetos para criar uma interface gráfica do usuário. Na realidade, XIB significa XML Interface Builder (Construtor de Interfaces XML).

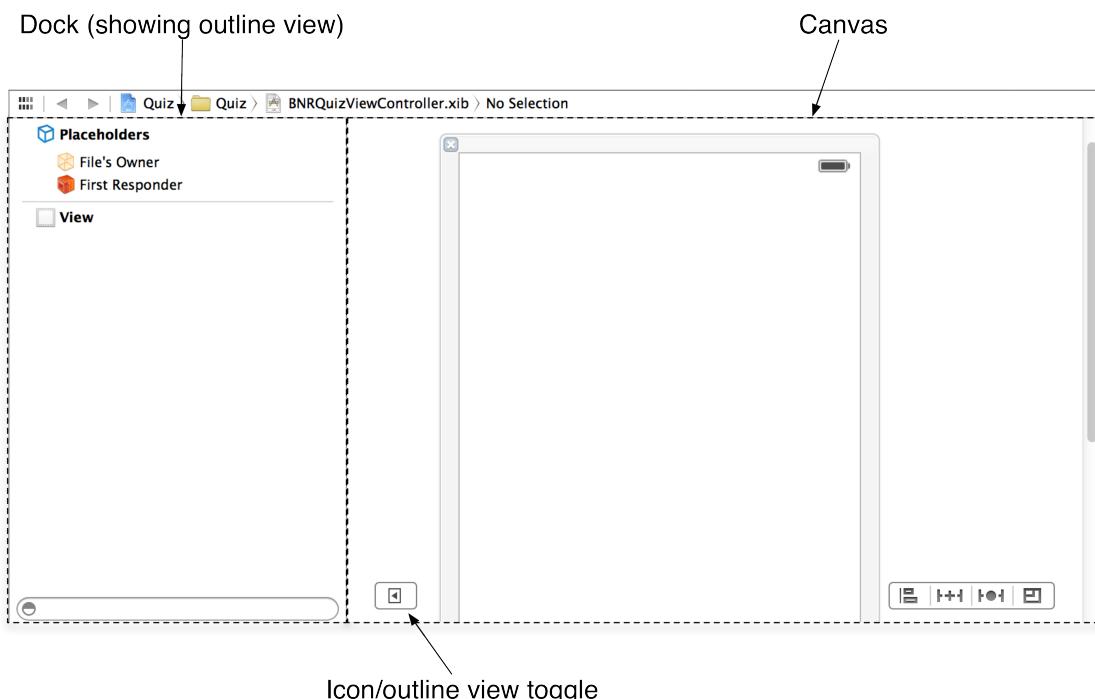
Em muitos construtores de GUI em outras plataformas, você descreve como quer que uma aplicação seja e então pressiona um botão para gerar código. O Interface Builder é diferente. Trata-se de um editor de objetos: você cria e configura objetos, como botões e tabelas, e depois os salva em um arquivo. Esse é o arquivo XIB.

O Interface Builder dividiu a área do editor em duas seções: o *dock* fica no lado esquerdo e o *canvas*, no lado direito.

O dock lista os objetos no arquivo XIB seja como ícones (*visão de ícones*) ou em palavras *visão de outline*). A visão de ícone é útil quando há pouco espaço na tela. Entretanto, para fins educacionais, é mais fácil ver o que está acontecendo na visão de outline.

Se o dock estiver na visão de ícones, clique no botão de revelação no canto inferior esquerdo do canvas para revelar a visão de outline (Figure 1.11).

Figure 1.11 Edição de um arquivo XIB no Interface Builder



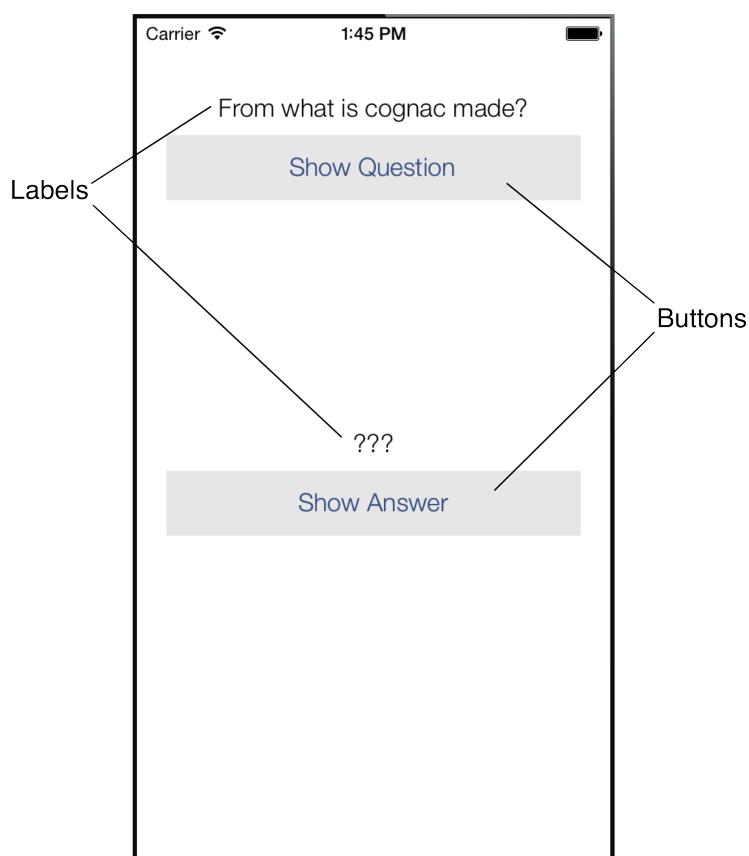
A visão de outline mostra que o `BNRQuizViewController.xib` contém três objetos: dois placeholders e um View. Ignore os placeholders por enquanto; você aprenderá mais sobre eles mais tarde.

O objeto View é uma instância de `UIView`. Esse objeto é a base da sua interface do usuário e você pode vê-lo no canvas. O canvas mostra como a sua interface do usuário aparecerá no aplicativo.

Clique no objeto View na estrutura do documento para selecioná-lo no canvas. Você pode mover a visão arrastando-a. Observe que movimentar a visão não muda nada em relação ao objeto propriamente dito; apenas reorganiza o canvas. Você também pode fechar a visão, clicando no x, no canto superior esquerdo. Novamente, isso não exclui a visão, apenas a remove do canvas. Você pode obtê-la de volta, selecionando-a novamente na visão de outline.

Por enquanto, a sua interface consiste em apenas esse objeto de visão. Você precisa adicionar mais quatro objetos de visão no Quiz: dois rótulos e dois botões.

Figure 1.12 Rótulos e botões necessários



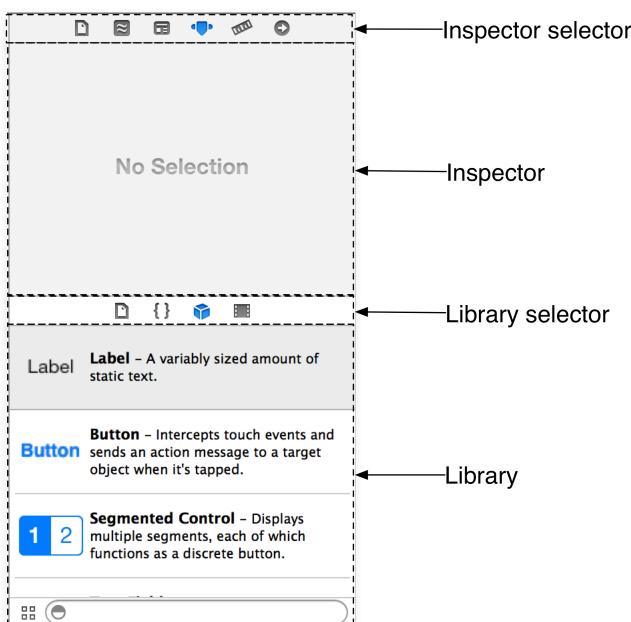
## Criação de objetos de visão

Para adicionar esses novos objetos, você deve ir até a biblioteca de objetos na *área de utilitários*.

A área de utilitários fica à direita da área do editor e tem duas seções: o *inspetor* e a *biblioteca*. A seção superior é o inspetor, que contém definições para o arquivo ou objeto selecionado na área do editor. A seção inferior é a biblioteca, que lista os itens que você pode adicionar a um arquivo ou projeto.

Na parte de cima de cada seção, há um seletor de diferentes tipos de inspetores e bibliotecas (Figure 1.13). No seletor de bibliotecas, selecione a aba para revelar a *biblioteca de objetos*.

Figure 1.13 Área de utilitários do Xcode



A biblioteca de objetos contém os objetos que você pode adicionar a um arquivo XIB para compor sua interface. Encontre o objeto Label. (Ele pode estar bem em cima; se não, role a lista para baixo ou use a barra de pesquisa na parte inferior da biblioteca.) Selecione esse objeto na biblioteca e arraste-o para o objeto de visão no canvas. Posicione o rótulo no centro da visão, próximo ao topo. Arraste um segundo rótulo para essa visão e posicione-o no centro, mais para baixo.

A seguir, localize Button na biblioteca de objetos e arraste dois botões para a visão. Posicione um embaixo de cada rótulo.

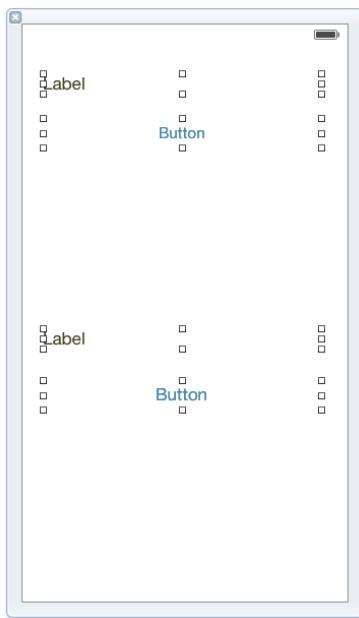
Agora você criou quatro objetos de visão e os adicionou à interface do usuário de `BNRQuizViewController`. Confirme na estrutura do documento.

## Configuração de objetos de visão

Agora que você criou os objetos de visão, você pode configurar os atributos deles. Alguns atributos, tais como tamanho, posição e texto, podem ser alterados diretamente no canvas. Outros devem ser alterados no *inspetor de atributos*, uma ferramenta que você usará em breve.

Você pode redimensionar um objeto selecionando-o no canvas ou na visão de outline e arrastando os cantos e bordas no canvas. Redimensione todos os quatro objetos de visão para que eles preencham a maior parte da janela.

Figure 1.14 Esticando os rótulos e botões



Você pode editar o título de um botão ou rótulo dando um clique duplo nele e digitando um texto novo. Altere o título do botão superior para Show Question e o do botão inferior para Show Answer. Altere o rótulo inferior para que ele mostre ??? . Exclua o texto no rótulo superior e deixe-o em branco. (No final, esse rótulo mostrará a pergunta para o usuário.) Sua interface deve ser semelhante à da Figure 1.15.

Figure 1.15 Definição de texto nos rótulos e botões

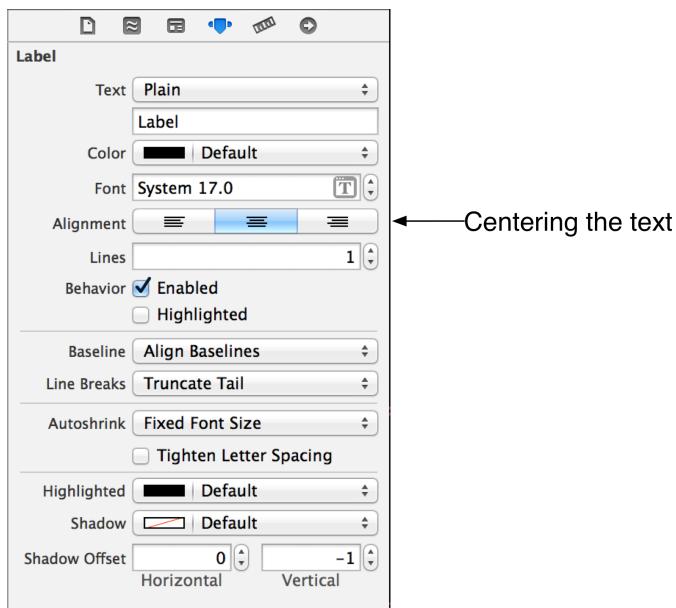


Seria melhor se o texto nos rótulos fosse centralizado. O alinhamento do texto de um rótulo deve ser definido no inspetor de atributos.

Selecione a guia para revelar o inspetor de atributos. Depois, selecione o rótulo inferior no canvas.

No inspetor de atributos, localize o controle segmentado de alinhamento. Selecione a opção de texto centralizado, conforme mostrado na Figure 1.16.

Figure 1.16 Centralização de texto do rótulo



Voltando ao canvas, observe que o ??? agora está centralizado no rótulo inferior. Selecione o rótulo superior no canvas e volte ao inspetor de atributos para definir o alinhamento do texto. (Esse rótulo não tem nenhum texto para mostrar agora, mas terá quando o aplicativo for executado.)

Para informar ao usuário onde ele pode clicar, você pode mudar a cor de fundo dos botões. Selecione o botão Show Question no canvas.

No inspetor de atributos, role para baixo até que você veja os atributos sob o cabeçalho View. Próximo ao rótulo Background, clique na cor (a caixa branca com uma barra vermelha). O seletor de cores aparecerá. Escolha uma cor que caia bem com o texto azul do botão.

Faça o mesmo no segundo botão, mas, em vez de clicar na cor no lado esquerdo, clique no lado direito, que tem texto e as setas para cima e para baixo. Será mostrada uma lista das últimas cores usadas em ordem cronológica e algumas cores padrão do sistema. Use esse seletor para escolher a mesma cor de fundo para o segundo botão.

## Arquivos NIB

A esta altura, você deve estar se perguntando como esses objetos ganham vida quando o aplicativo é executado.

Quando você constrói um aplicativo que utiliza um arquivo XIB, o arquivo XIB é compilado em um arquivo NIB, que é mais leve e mais fácil de ser analisado pelo aplicativo. Então, o arquivo NIB é copiado para o *pacote* do aplicativo. O pacote é um diretório que contém o executável do aplicativo e todos os recursos usados por esse executável.

No tempo de execução, o aplicativo lerá, ou carregará, o arquivo NIB quando sua interface for necessária. O Quiz tem apenas um arquivo XIB e, assim, terá apenas um arquivo NIB no pacote. O arquivo NIB único do Quiz será carregado na primeira vez em que o aplicativo for aberto. Um aplicativo complexo, no entanto, pode ter muitos arquivos NIB que são carregados conforme necessário. Você aprenderá mais sobre o carregamento de arquivos NIB no Chapter 6.

A interface do seu aplicativo agora tem a aparência certa. Mas, para que ela se torne funcional, você deve fazer algumas conexões entre esses objetos de visão e a `BNRQuizViewController` que comandará o espetáculo.

## Fazendo conexões

Uma *conexão* permite que um objeto saiba a localização de outro objeto na memória para que eles possam se comunicar. Você pode fazer dois tipos de conexão no Interface Builder: outlets e ações. Um *outlet* aponta para um objeto. (Se você não estiver familiarizado com “ponteiros”, você aprenderá mais sobre eles no Chapter 2.)

Uma *ação* é um método que é ativado por um botão ou alguma outra visão com a qual o usuário possa interagir, como um controle deslizante ou um seletor.

Vamos começar criando outlets que apontam para as instâncias de **UILabel**. É hora de sair um pouco do Interface Builder e escrever algum código.

## Declaração de outlets

No navegador de projetos, encontre e selecione o arquivo chamado `BNRQuizViewController.m`. A área do editor mudará do Interface Builder para o editor de códigos do Xcode.

No `BNRQuizViewController.m`, exclua qualquer código que o template tenha adicionado entre as diretivas `@implementation` e `@end`, de forma que o arquivo fique semelhante ao seguinte:

```
#import "BNRQuizViewController.h"

@interface BNRQuizViewController ()

@end

@implementation BNRQuizViewController

@end
```

A seguir, adicione o código a seguir. Não se preocupe se não entendê-lo por enquanto; basta colocá-lo lá.

```
#import "BNRQuizViewController.h"

@interface BNRQuizViewController ()

@property (nonatomic, weak) IBOutlet UILabel *questionLabel;
@property (nonatomic, weak) IBOutlet UILabel *answerLabel;

@end

@implementation

@end
```

Você notou o negrito? Neste livro, os códigos que você deve digitar estarão sempre em negrito; o código que não estiver em negrito fornece o contexto onde adicionar códigos novos.

Neste novo código, você declarou duas propriedades. Você aprenderá mais sobre propriedades no Chapter 3. Por enquanto, concentre-se na segunda metade da primeira linha.

```
@property (nonatomic, weak) IBOutlet UILabel *questionLabel;
```

Esse código dá a todas as instâncias de **BNRViewController** um outlet chamado `questionLabel`, que pode ser usado para apontar para um objeto **UILabel**. A palavra-chave `IBOutlet` comunica ao Xcode que você definirá esse outlet utilizando o Interface Builder.

## Definição de outlets

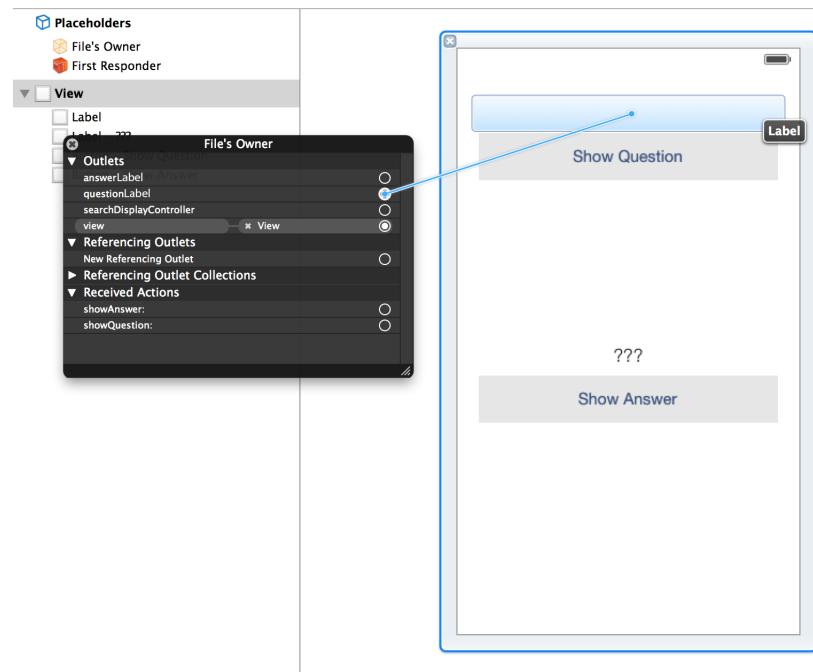
No navegador de projetos, selecione `BNRQuizViewController.xib` para reabrir o Interface Builder.

Você quer que o outlet `questionLabel` aponte para a instância de **UILabel** no topo da interface do usuário.

No dock, localize a seção **Placeholders** e o objeto **File's Owner**. Um placeholder substitui outro objeto em um aplicativo em execução. No seu caso, o **File's Owner** substitui uma instância de **BNRQuizViewController**, que é o objeto responsável pelo gerenciamento da interface definida no `BNRQuizViewController.xib`.

No dock, clique com o botão direito ou clique+Control no **File's Owner** para exibir o painel de conexões (Figure 1.17). Depois, arraste do círculo ao lado de `questionLabel` para a **UILabel**. Quando o rótulo estiver realçado, solte o botão do mouse e o outlet será definido.

Figure 1.17 Definição de questionLabel

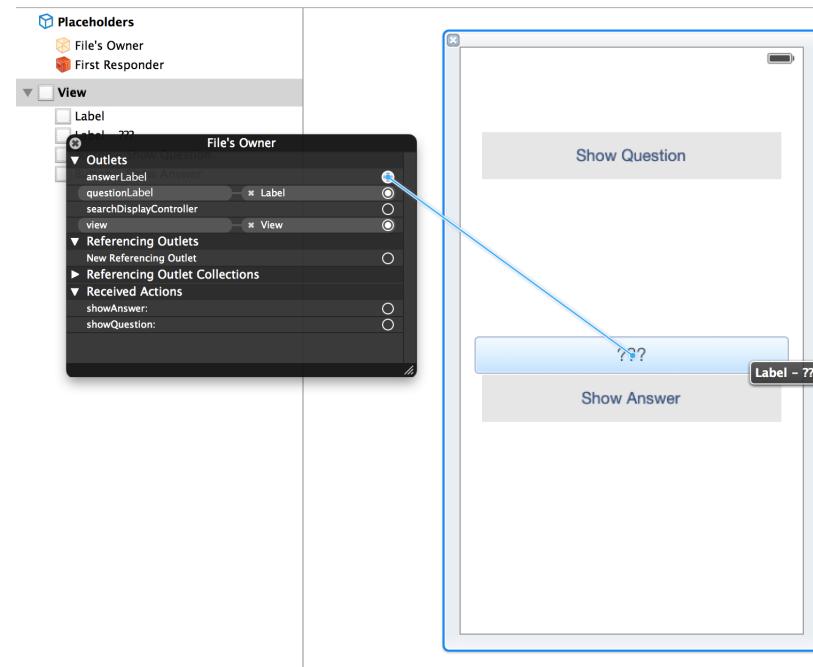


(Se você não encontrar `questionLabel` no painel de conexões, revise o seu arquivo `BNRQuizViewController.m` para verificar se há algum erro de digitação.)

Agora, quando o arquivo NIB for carregado, o outlet `questionLabel` de `BNRQuizViewController` apontará automaticamente para a instância de `UILabel` na parte superior da tela. Isso permitirá que `BNRQuizViewController` comunique ao rótulo qual questão mostrar.

Configure o outlet `answerLabel` da mesma maneira: arraste do círculo ao lado de `answerLabel` para a `UILabel` inferior (Figure 1.18).

Figure 1.18 Definição de answerLabel



Observe que você arrasta *a partir* do objeto com o outlet que você quer definir *para* o objeto para o qual você quer que o outlet aponte.

Seus outlets estão todos configurados. As próximas conexões que você precisa fazer envolvem os dois botões.

Quando um **UIButton** é tocado, ele envia uma mensagem para outro objeto. O objeto que recebe a mensagem é chamado de *destino*. A mensagem que é enviada é chamada de *ação*. Essa ação é o nome do método que contém o código a ser executado em resposta ao toque no botão.

No seu aplicativo, o destino de ambos os botões será a instância de **BNRQuizViewController**. Cada botão terá sua própria ação. Vamos começar definindo os dois métodos de ação: **showQuestion:** e **showAnswer:**

## Definição de métodos de ação

Volte para o **BNRQuizViewController.m** e adicione o seguinte código entre **@implementation** e **@end**.

```
@implementation

- (IBAction)showQuestion:(id)sender
{
}

- (IBAction)showAnswer:(id)sender
{
}

@end
```

Você dará corpo a esses métodos após ter feito as conexões entre destino e ação. A palavra-chave **IBAction** comunica ao Xcode que você fará tais conexões no Interface Builder.

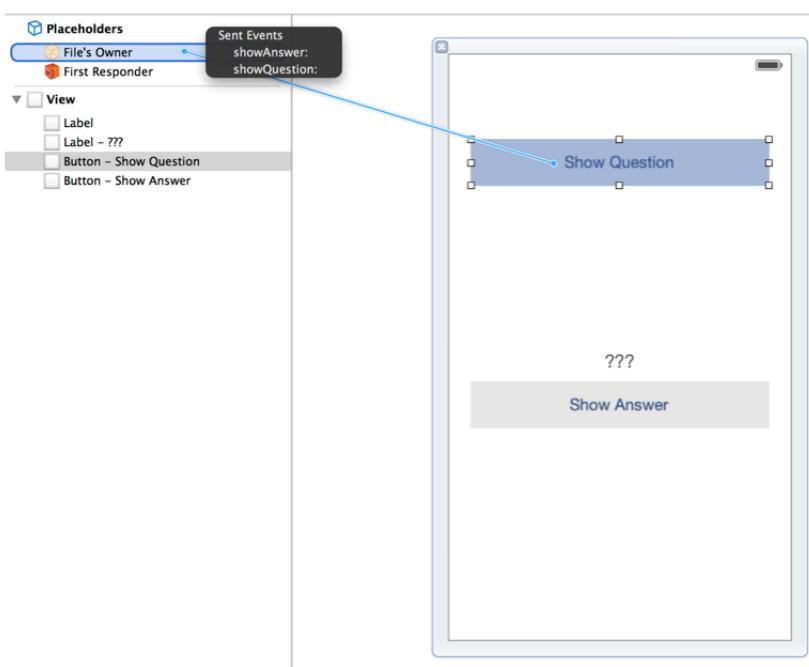
## Definição de destinos e ações

Para definir o destino de um objeto, segure a tecla Control e arraste *do* objeto *para* o destino dele. Quando você soltar o mouse, o destino terá sido definido, e um menu pop-up permitirá que você escolha a ação.

Vamos começar com o botão Show Question. O destino deve ser **BNRQuizViewController** e a sua ação, **showQuestion:**.

Abra novamente o **BNRQuizViewController.xib**. Selecione o botão Show Question no canvas e pressione Control+arraste (ou clique com o botão direito+arraste) para o File's Owner. Quando o File's Owner estiver realçado, solte o botão do mouse e selecione **showQuestion:** no menu pop-up, conforme mostrado na Figure 1.19.

Figure 1.19 Definição de destino/ação para Show Question



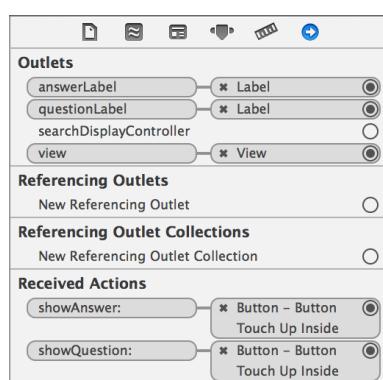
Agora iremos para o botão Show Answer. Selecione o botão e pressione Control+arraste do botão para o File's Owner. Depois, selecione **showAnswer:** no menu pop-up.

## Resumo das conexões

Agora há cinco conexões entre o seu **BNRQuizViewController** e os objetos de visão. Você configurou os ponteiros `answerLabel` e `questionLabel` para apontar para os rótulos – duas conexões. **BNRQuizViewController** é o alvo de ambos os botões – mais duas conexões. O template do projeto fez mais uma conexão: o ponteiro `view` de **BNRQuizViewController** está conectado ao objeto View que representa o fundo do aplicativo. Agora são cinco.

Você pode verificar essas conexões no *inspetor de conexões*. Selecione o File's Owner na visão de outline. Depois, no inspetor, clique na guia para revelar o inspetor de conexões. (Figure 1.20).

Figure 1.20 Verificação de conexões no inspetor



Seu arquivo XIB está completo. Os objetos de visão foram criados e configurados, e todas as conexões necessárias com o objeto de controlador foram feitas. Agora, vamos criar e conectar seus objetos de modelo.

## Criação de objetos de modelo

Objetos de visão compõem a interface do usuário, para que os desenvolvedores normalmente criem, configurem e conectem objetos de visão utilizando o Interface Builder. Objetos de modelo, por outro lado, são configurados via código.

No navegador de projetos, selecione `BNRQuizViewController.m`. Adicione o seguinte código que declara um inteiro e ponteiros para dois arrays.

```
@interface BNRQuizViewController ()  
  
@property (nonatomic) int currentQuestionIndex;  
  
@property (nonatomic, copy) NSArray *questions;  
@property (nonatomic, copy) NSArray *answers;  
  
@property (nonatomic, weak) IBOutlet UILabel *questionLabel;  
@property (nonatomic, weak) IBOutlet UILabel *answerLabel;  
  
@end  
  
@implementation  
  
@end
```

Os arrays serão listas ordenadas contendo perguntas e respostas. O inteiro acompanhará em qual pergunta o usuário está.

Esses arrays devem estar prontos ao mesmo tempo em que a interface aparece para o usuário. Para garantir que isso aconteça, você criará os arrays logo depois da criação de uma instância de **BNRQuizViewController**.

Quando uma instância de **BNRQuizViewController** é criada, é enviada a ela a mensagem **initWithNibName:bundle:**. No **BNRQuizViewController.m**, implemente o método **initWithNibName:bundle:**.

```
...

@property (nonatomic, weak) IBOutlet UILabel *answerLabel;

@end

@implementation BNRQuizViewController

- (instancetype)initWithNibName:(NSString *)nibNameOrNil
                           bundle:(NSBundle *)nibBundleOrNil
{
    // Call the init method implemented by the superclass
    self = [super initWithNibName:nibNameOrNil bundle:nibBundleOrNil];

    if (self) {
        // Create two arrays filled with questions and answers
        // and make the pointers point to them

        self.questions = @[@"From what is cognac made?",
                           @"What is 7+7?",
                           @"What is the capital of Vermont?"];

        self.answers = @[@"Grapes",
                         @"14",
                         @"Montpelier"];
    }

    // Return the address of the new object
    return self;
}

@end
```

(Sintaxe assustadora? Está com medo? Não entre em pânico – você aprenderá mais sobre a linguagem Objective-C nos próximos dois capítulos.)

## Utilização de preenchimento de código

Conforme você for seguindo neste livro, você digitará muito código. Observe que enquanto você digitava esse código, o Xcode estava pronto para preencher partes dele para você. Por exemplo, quando você começou a digitar **initWithNibName:bundle:**, ele sugeriu esse método antes de você terminar de digitar. Você pode pressionar a tecla Enter para aceitar a sugestão do Xcode ou selecionar outra sugestão da caixa pop-up exibida.

Quando você aceita uma sugestão de preenchimento de código para um método que recebe argumentos, o Xcode coloca *placeholders* nas áreas dos argumentos. (Perceba que esse uso do termo “placeholder” é completamente diferente dos objetos placeholder que você viu no arquivo XIB.)

Placeholders não são códigos válidos, e você precisa substituí-los para compilar seu aplicativo. Isso pode ficar confuso, porque os placeholders frequentemente têm os mesmos nomes que você quer dar aos argumentos. Portanto, o texto do código parece estar totalmente correto, mas ocorre um erro.

A Figure 1.21 mostra dois placeholders que você deve ter visto ao digitar o código anterior.

Figure 1.21 Placeholders no preenchimento de códigos e erros

Placeholders: select and press Return to replace with arguments of the same names

The screenshot shows the Xcode interface with the file BNRQuizViewController.m open. The code is as follows:

```

16 @implementation BNRQuizViewController
17
18 - (id)initWithNibName:(NSString *)NibNameOrNil bundle:(NSBundle *)bundleOrNil
19 {
20     self = [super initWithNibName:NibNameOrNil bundle:bundleOrNil];
21
22     if (self) {
23         self.questions = @{@"From what is cognac made?", @"What is 7+7?", @"What is the capital of Vermont?"};
24
25         self.answers = @{@"Grapes", @"14", @"Montpelier"};
26     }
27
28     return self;
29 }
30
31 @end

```

Two red circles highlight placeholder text: 'NibNameOrNil' at line 18 and 'bundleOrNil' at line 19. Lines from the 'Placeholders...' caption point to these circled areas.

Você está vendo o `nibNameOrNil` e o `nibBundleOrNil` na primeira linha da implementação do `initWithNibName:bundle:`? Eles são placeholders. Você pode perceber porque eles estão dentro de retângulos arredondados levemente sombreados. A solução é selecionar os placeholders, digitar o nome correto do argumento e pressionar a tecla Enter. Os retângulos arredondados desaparecem, e o seu código estará correto e válido.

Como nesse caso os placeholders têm o texto correto, você pode simplesmente selecionar um placeholder e pressionar Enter para que o Xcode o substitua por um argumento do mesmo nome.

Quando usar o preenchimento de código, fique atento a nomes que são *parecidos* com os que você quer. Cocoa Touch usa convenções de nomenclatura que frequentemente fazem com que métodos, tipos e variáveis distintas tenham nomes muito parecidos. Assim, não aceite impulsivamente a primeira sugestão que o Xcode oferecer sem verificar a. Sempre verifique com atenção.

## Juntando tudo

Você criou, configurou e conectou os objetos de visão e seu controlador de visão. Você criou os objetos de modelo. Falta fazer duas coisas no Quiz:

- terminar de implementar os métodos de ação `showQuestion:` e `showAnswer:` no `BNRQuizViewController`
- adicionar ao `BNRAppDelegate` um código que criará e mostrará uma instância de `BNRQuizViewController`

## Implementação de métodos de ação

No `BNRQuizViewController.m`, termine as implementações do `showQuestion:` e do `showAnswer:`.

```
    ...
    // Return the address of the new object
    return self;
}

- (IBAction)showQuestion:(id)sender
{
    // Step to the next question
    self.currentQuestionIndex++;

    // Am I past the last question?
    if (self.currentQuestionIndex == [self.questions count]) {

        // Go back to the first question
        self.currentQuestionIndex = 0;
    }

    // Get the string at that index in the questions array
    NSString *question = self.questions[self.currentQuestionIndex];

    // Display the string in the question label
    self.questionLabel.text = question;

    // Reset the answer label
    self.answerLabel.text = @"????";
}

- (IBAction)showAnswer:(id)sender
{
    // What is the answer to the current question?
    NSString *answer = self.answers[self.currentQuestionIndex];

    // Display it in the answer label
    self.answerLabel.text = answer;
}

@end
```

## Colocando o controlador de visão na tela

Se você fosse executar o Quiz agora, não veria a interface que você criou no `BNRQuizViewController.xib`. Ao invés disso, você veria uma tela em branco. Para que a sua interface seja colocada na tela, você deve conectar seu controlador de visão a outro controlador do aplicativo: o `BNRAppDelegate`.

Sempre que você cria um aplicativo iOS utilizando um template Xcode, um *delegate de aplicativo* é criado para você. O delegate é o ponto inicial de um aplicativo, e todo aplicativo para iOS tem um.

Ele gerencia uma única `UIWindow` de alto nível para o aplicativo. Para que o `BNRQuizViewController` apareça na tela, você precisa torná-lo o *controlador de visão raiz* da janela.

Quando um aplicativo iOS é iniciado, não está imediatamente pronto para o usuário. Há um pouco de configuração por trás dos panos. Logo antes do aplicativo estar pronto para o usuário, o delegate do aplicativo recebe a mensagem `application:didFinishLaunchingWithOptions:`. Essa é a sua chance de preparar o aplicativo para a ação. Em particular, você deve garantir que a sua interface esteja pronta antes de o usuário ter a chance de interagir com ela.

No navegador de projetos, localize e selecione `BNRAppDelegate.m`. Adicione o seguinte código ao método `application:didFinishLaunchingWithOptions:` para criar uma instância de `BNRQuizViewController` e para configurá-lo como o controlador de visão raiz da janela do delegate do aplicativo.

```
#import "BNRAppDelegate.h"
#import "BNRQuizViewController.h"

- (BOOL)application:(UIApplication *)application
    didFinishLaunchingWithOptions:(NSDictionary *)launchOptions
{
    self.window = [[UIWindow alloc] initWithFrame:[[UIScreen mainScreen] bounds]];
    // Override point for customization after application launch

    BNRQuizViewController *quizVC = [[BNRQuizViewController alloc] init];
    self.window.rootViewController = quizVC;

    self.window.backgroundColor = [UIColor whiteColor];
    [self.window makeKeyAndVisible];
    return YES;
}
```

Agora, toda vez que o aplicativo for iniciado, uma instância de **BNRQuizViewController** será criada. Depois, a instância receberá a mensagem **initWithNibName:bundle:**, que ativará o carregamento do arquivo NIB do **BNRQuizViewController.xib** e a criação dos objetos de modelo.

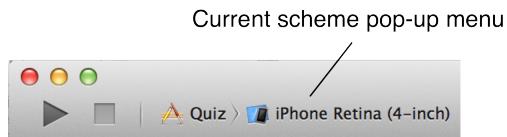
O seu aplicativo Quiz está pronto. É hora de testá-lo.

## Execução no simulador

Primeiro, você irá executar o Quiz no simulador de iOS do Xcode. Depois, você verá como executá-lo em um dispositivo real.

Para preparar o Quiz para ser executado no simulador, localize o menu pop-up do esquema atual na barra de ferramentas do Xcode (Figure 1.22).

Figure 1.22 Esquema iPhone Retina (4-inch) selecionado



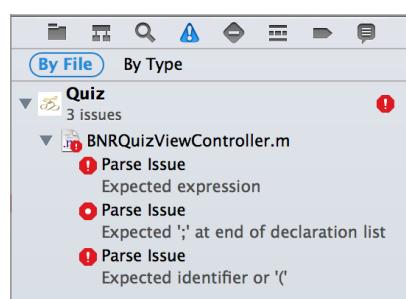
Se ele informar algo genérico, como iPhone Retina (4-inch), então o projeto está configurado para ser executado no simulador e você pode prosseguir. Se informar algo como Christian's iPhone, então clique nele e selecione iPhone Retina (4-inch) no menu pop-up.

Para este livro, use o esquema iPhone Retina (4-inch). A única diferença entre ele e o esquema iPhone Retina (3.5-inch) é a altura da tela. Se você selecionar o simulador de 3,5 polegadas, partes de sua interface do usuário podem ser cortadas. Discutiremos como redimensionar as interfaces para ambos os tamanhos de tela do iPhone (e do iPad também) no Chapter 15.

A seguir, clique no botão de reprodução parecido com o do iTunes na barra de ferramentas. Isso construirá (compilará) e executará o aplicativo. Você fará essa ação tantas vezes que talvez seja melhor aprender e usar o atalho de teclado Command-R.

Se a compilação retornar quaisquer erros, você pode visualizá-los no *navegador de problemas* selecionando a guia **⚠** na área de navegação (Figure 1.23).

Figure 1.23 Navegador de problemas com erros e avisos de exemplo



Você pode clicar em qualquer erro ou aviso no navegador de problemas para ser levado ao arquivo e à linha de código onde ocorreu o problema. Localize e corrija todos os problemas existentes (isto é, erros de digitação no código!) comparando o seu código com o do livro. Depois, tente executar novamente o aplicativo. Repita este processo até que o seu aplicativo seja compilado.

Uma vez que seu aplicativo tenha sido compilado, ele será iniciado no simulador de iOS. Se for a primeira vez que você está usando o simulador, pode demorar um pouco para o aplicativo Quiz aparecer.

Brinque um pouco com o aplicativo Quiz. Você deve ser capaz de tocar no botão Show Question para ver uma nova pergunta no rótulo superior; tocando novamente em Show Answer, você deve ver a resposta. Se o seu aplicativo não estiver funcionando da forma esperada, revise as conexões no `BNRQuizViewController.xib`.

## Implementação de um aplicativo

Agora que você escreveu seu primeiro aplicativo para iOS e o executou em um simulador, é a hora de implementá-lo em um dispositivo.

Para instalar um aplicativo no seu dispositivo de desenvolvimento, você precisa de um certificado de desenvolvedor da Apple. Os certificados de desenvolvedor são emitidos para Desenvolvedores de iOS registrados que pagaram a taxa de desenvolvedor. Esse certificado permite que você assine o seu código e possa executá-lo em um dispositivo. Sem um certificado válido, os dispositivos não rodam o seu aplicativo.

O Apple's Developer Program Portal (<http://developer.apple.com>) contém todas as instruções e recursos para a obtenção de um certificado válido. A interface para o processo de set-up é constantemente atualizada pela Apple e, portanto, seria perda de tempo descrevê-la em detalhes. Em vez disso, visite nosso guia em [http://www.bignerdranch.com/ios\\_device\\_provisioning](http://www.bignerdranch.com/ios_device_provisioning) para obter instruções.

Se você está curioso sobre o que exatamente está acontecendo aqui, há quatro itens importantes no processo de provisionamento:

|                              |                                                                                                                                                                                                                                                                                                                                                                                                                                   |
|------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Certificado de desenvolvedor | Este certificado é adicionado ao seu keychain do Mac usando Keychain Access. Ele é usado para assinar digitalmente o seu código.                                                                                                                                                                                                                                                                                                  |
| App ID                       | O identificador de aplicativo é uma string que identifica exclusivamente o seu aplicativo na App Store. Os identificadores de aplicativos geralmente são semelhantes ao seguinte: <code>com.bignerdranchAwesomeApp</code> , com o nome do aplicativo logo após o nome da sua empresa.                                                                                                                                             |
| Device ID (UDID)             | O App ID no seu perfil de provisionamento deve corresponder ao <i>identificador de pacote</i> do seu aplicativo. Um perfil de desenvolvimento pode ter um caractere curinga (*) para o respectivo App ID e, portanto, corresponderá a qualquer identificador de pacote. Para ver o identificador de pacote do aplicativo Quiz, selecione o projeto no navegador de projetos. Depois, selecione o destino Quiz e o painel General. |
| Perfil de provisionamento    | Identificador único para cada dispositivo iOS.                                                                                                                                                                                                                                                                                                                                                                                    |
| Período de provisionamento   | Arquivo que reside no seu dispositivo de desenvolvimento e no seu computador. Ele faz referência a um Certificado de Desenvolvedor, a um único App ID e a uma lista de Device IDs para os dispositivos nos quais o aplicativo pode ser instalado. Esse arquivo possui o sufixo <code>.mobileprovision</code> .                                                                                                                    |

Quando um aplicativo é implementado em um dispositivo, o Xcode usa um perfil de provisionamento no seu computador para acessar o certificado correto. Esse certificado é usado para assinar o binário do aplicativo. Depois, o UDID do dispositivo de desenvolvimento é comparado a um dos UDIDs contidos no perfil de provisionamento, e o App ID é correspondido com o identificador de pacote. O binário assinado é então enviado ao seu dispositivo de desenvolvimento, onde é confirmado pelo mesmo perfil de provisionamento no dispositivo e, finalmente, inicializado.

Abra o Xcode e conecte o seu dispositivo de desenvolvimento (iPhone, iPod touch ou iPad) ao seu computador. Isso deve abrir automaticamente a janela Organizer, que você pode reabrir a qualquer momento utilizando o

item Organizer no menu Window. Na janela Organizer, você pode selecionar a guia Devices para ver toda a informação de provisionamento.

Para rodar o aplicativo Quiz no seu dispositivo, você precisa dizer ao Xcode para implementar no dispositivo e não no simulador. Volte à descrição do esquema atual na barra de ferramentas do Xcode. Clique na descrição e, em seguida, selecione iOS Device no menu pop-up (Figure 1.24). Se não houver a opção iOS Device, encontre a opção mais parecida com Christian's iPhone.

Figure 1.24 Escolhendo o dispositivo



Compile e execute seu aplicativo (Command-R) e ele aparecerá no seu dispositivo.

## Ícones de aplicativo

Ao rodar o aplicativo Quiz (em seu dispositivo de desenvolvimento ou no simulador), retorno para a tela Home (Início) do dispositivo. Você verá que o ícone do aplicativo é um bloco padrão e desinteressante. Vamos dar ao Quiz um ícone melhor.

Um *ícone de aplicativo* é uma imagem simples que representa o aplicativo na tela inicial do iOS. Os diferentes dispositivos requerem ícones de tamanhos diferentes, e esses requisitos são mostrados na Table 1.1.

Table 1.1 Tamanhos de ícones de aplicativos por dispositivo

| Dispositivo                            | Tamanhos de ícones de aplicativos     |
|----------------------------------------|---------------------------------------|
| iPhone/iPod touch (iOS 7)              | 120x120 pixels (a 2x)                 |
| iPhone/iPod touch (iOS 6 e anteriores) | 57x57 pixels<br>114x114 pixels (a 2x) |
| iPad (iOS 7 e anteriores)              | 72x72 pixels<br>144x144 pixels (a 2x) |

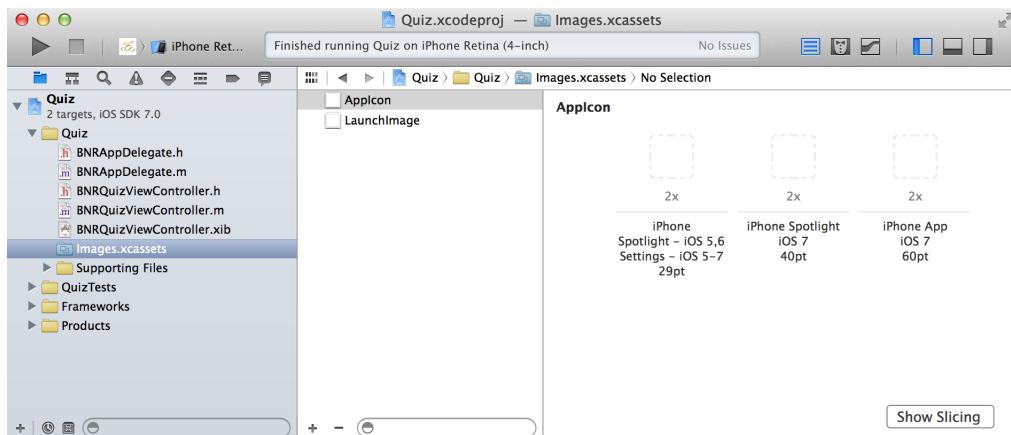
Todo aplicativo que você implementar na App Store deve ter ícones para todas as classes de dispositivos nas quais ele pode ser executado. Por exemplo, se você der suporte apenas para iPhone e iPod touch com iOS 7 ou posteriores, então você só precisa fornecer uma imagem (com a resolução listada acima). No extremo oposto, se você tem um aplicativo universal compatível com iOS 6 e versões posteriores, você terá que ter cinco ícones de aplicativos com diferentes resoluções, as duas para o iPad e as três outras para o iPhone e o iPod touch.

Preparamos um arquivo de imagem de ícone (tamanho 120x120) para o aplicativo Quiz. Você pode baixar esse ícone (e outros recursos dos outros capítulos) em <http://www.bignerdranch.com/solutions/iOSProgramming4ed.zip>. Descompacte o iOSProgramming4ed.zip e localize o arquivo Icon@2x.png no diretório Resources da pasta descompactada.

Você adicionará esse ícone ao seu pacote de aplicativo como um *recurso*. Geralmente, há dois tipos de arquivos em um aplicativo: código e recursos. O código (como BNRQuizViewController.h e BNRQuizViewController.m) é usado para criar o aplicativo propriamente dito. Os recursos incluem coisas como imagens e sons que são usados pelo aplicativo no tempo de execução. Arquivos XIB, que são compilados em arquivos NIB lidos no tempo de execução, também são recursos.

No navegador de projetos, localize Images.xcassets. Selecione esse arquivo para abri-lo e então selecione AppIcon na lista de recursos do lado esquerdo.

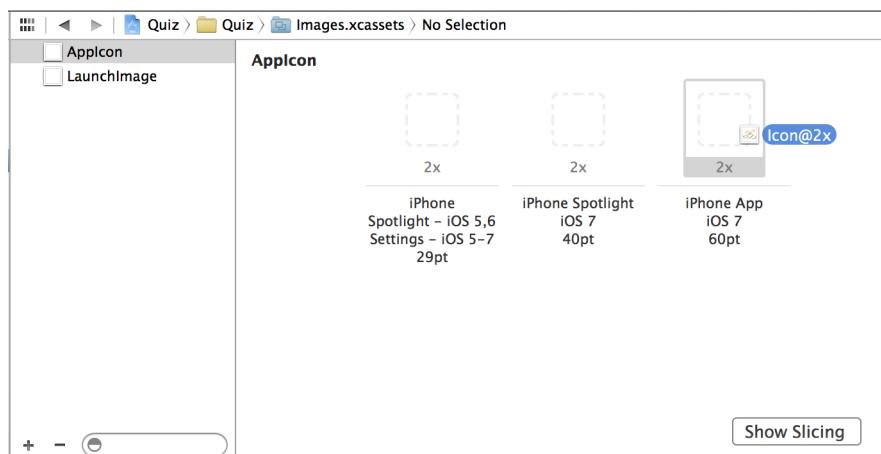
Figure 1.25 Exibindo o Asset Catalog (catálogo de ativos)



Esse painel é o *Asset Catalog* (catálogo de ativos), onde você pode gerenciar todas as imagens de que o seu aplicativo precisará.

Arraste o arquivo `Icon@2x.png` do Finder para as margens da seção `AppIcon`. Isso copiará o arquivo para o diretório do seu projeto no sistema de arquivos, e adicionará uma referência a esse arquivo no Asset Catalog. (Você pode pressionar Control e clicar em um arquivo no Asset Catalog e selecionar a opção `Show in Finder` para confirmar.)

Figure 1.26 Adição do ícone do aplicativo ao Asset Catalog (catálogo de ativos)



Compile e execute o aplicativo novamente. Saia do aplicativo e procure o aplicativo Quiz com o logotipo BNR.

(Se você não enxergar o ícone, exclua o aplicativo e então compile-o e execute-o novamente para reimplementá-lo. Em um dispositivo, faça o mesmo que faria com qualquer outro aplicativo. A opção mais simples no simulador é reiniciá-lo. Com o simulador aberto, localize a barra de menus. Selecione `iOS Simulator` e, em seguida, `Reset Content and Settings....` Isso removerá todos os aplicativos e reinicializará o simulador para as configurações padrão. Você deverá ver o ícone do aplicativo da próxima vez que executá-lo.)

## Imagens de abertura

Outro recurso que você pode gerenciar no catálogo de ativos é a imagem de abertura do aplicativo, que aparece enquanto ele está carregando. A imagem de abertura tem um papel específico no iOS: ela informa ao usuário que o aplicativo está realmente carregando, e dá uma prévia da interface de usuário com a qual o usuário vai interagir depois que o aplicativo terminar de carregar. Portanto, uma boa imagem de abertura seria um instantâneo do aplicativo sem nenhum conteúdo. Por exemplo, a imagem de abertura do aplicativo *Clock* mostra as quatro guias na parte inferior, nenhuma delas selecionada. Uma vez que o aplicativo é carregado, a guia correta é selecionada e o conteúdo fica visível. (Lembre-se de que a imagem de abertura é substituída depois que o aplicativo é iniciado; ela não se torna a imagem de fundo do aplicativo.)

No diretório Resources, onde você encontra os ícones do aplicativo, há duas imagens de abertura: Default@2x.png e Default-568h@2x.png. Abra o item LaunchImage no Asset Catalog (catálogo de ativos) e depois arraste essas imagens para ele do mesmo modo que você fez com os ícones do aplicativo.

Compile e execute o aplicativo. Enquanto o aplicativo inicia, você verá por um breve período de tempo a imagem de abertura.

Por que duas imagens de abertura? Uma imagem de abertura deve caber na tela do dispositivo no qual o aplicativo está sendo iniciado. Assim, você precisa de uma imagem de abertura para telas Retina de 3,5 polegadas e outra para telas Retina de 4 polegadas. Observe que, se você pretende que o aplicativo seja compatível com iOS6 ou versões anteriores no iPhone e no iPod touch, você deve incluir uma terceira imagem de abertura não-Retina. A Table 1.2 mostra os diferentes tamanhos de imagem que você necessitará para cada tipo de dispositivo.

Table 1.2 Tamanho das imagens de abertura por dispositivo

| Dispositivo                                       | Tamanho da imagem de abertura                             |
|---------------------------------------------------|-----------------------------------------------------------|
| iPhone/iPod touch sem tela Retina                 | 320x480 pixels (apenas Retrato)                           |
| iPhone/iPod touch com tela Retina (3,5 polegadas) | 640x960 pixels (apenas Retrato)                           |
| iPhone/iPod touch com tela Retina (4 polegadas)   | 640x1136 pixels (apenas Retrato)                          |
| iPad sem tela Retina                              | 768x1024 pixels (Retrato)<br>1024x768 pixels (Paisagem)   |
| iPad com tela Retina                              | 1536x2048 pixels (Retrato)<br>2048x1536 pixels (Paisagem) |

(Note que a Table 1.2 lista as resoluções das telas dos dispositivos; a barra de status real é colocada sobre a imagem de abertura.)

Parabéns! Você escreveu seu primeiro aplicativo e o instalou no seu dispositivo. Agora é hora de explorar as grandes ideias que o fazem funcionar.



# 2

# Objective-C

Os aplicativos iOS são escritos na linguagem Objective-C usando os frameworks Cocoa Touch. Objective-C é uma extensão da linguagem C, e os frameworks Cocoa Touch são coleções de classes Objective-C.

Neste capítulo, você aprenderá princípios básicos de Objective-C e criará um aplicativo chamado `RandomItems`. Mesmo que você já conheça Objective-C, você deve ler este capítulo para criar a classe `BNRItem`, que será usada mais tarde no livro.

Este livro pressupõe que você tenha algum conhecimento de C e compreenda as ideias básicas da programação orientada a objetos. Se C ou programação orientada a objetos deixam você apreensivo, recomendamos que comece com o guia *Objective-C Programming: The Big Nerd Ranch Guide* (Programação em Objective-C: Guia da Big Nerd Ranch).

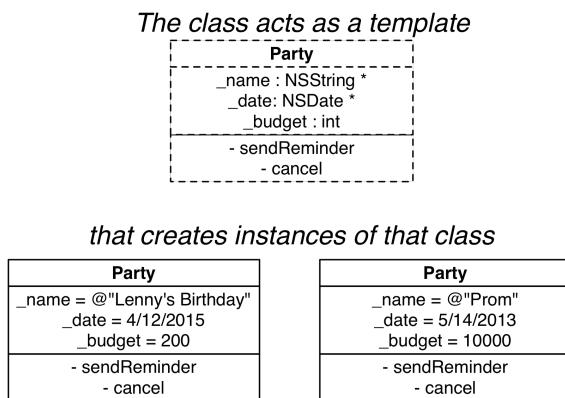
## Objetos

Digamos que você precise de uma maneira para representar uma festa. Uma festa tem alguns atributos exclusivos, como nome, data e uma lista de convidados. Você também pode pedir que a festa faça alguma coisa, como enviar um lembrete por e-mail a todos os convidados, imprimir crachás com nomes ou mesmo cancelar a festa.

No C, você definiria uma estrutura para armazenar os dados que descrevessem a festa. A estrutura teria membros de dados – um para cada atributo da festa. Cada membro de dados teria um nome e um tipo. Para criar uma festa específica, você usaria a função `malloc` para alocar um pedaço (chunk) de memória suficientemente grande para armazenar a estrutura.

No Objective-C, em vez de usar uma estrutura para representar a festa, você usa uma *classe*. A classe é como uma fôrma de biscoitos que produz objetos. A classe `Party` cria objetos, e esses objetos são instâncias da classe `Party`. Cada instância da classe `Party` armazena os dados para uma única festa (Figure 2.1).

Figure 2.1 Uma classe e suas instâncias



Uma instância de `Party`, assim como qualquer objeto, é um pedaço (chunk) de dados na memória, e armazena os valores dos seus atributos nas *variáveis de instância*. (Você também pode ver esses chamados de “ivars” em alguns lugares.) No Objective-C, geralmente colocamos um sublinhado no início do nome da variável de instância. Portanto, uma instância de `Party` pode ter como variáveis de instância `_name`, `_date` e `_budget`.

Uma estrutura C é um pedaço de memória, e um objeto é um pedaço de memória. Uma estrutura C tem membros de dados, cada qual com um nome e um tipo. Da mesma forma, um objeto tem variáveis de instância, cada qual com um nome e um tipo.

A diferença importante entre uma estrutura C e uma classe Objective-C é que uma classe tem *métodos*. O método é semelhante a uma função: tem um nome, um tipo de retorno e uma lista de parâmetros esperados por ele. Um método também tem acesso às variáveis de instância do objeto. Se você quer que um objeto execute o código de um de seus métodos, você envia para o objeto uma *mensagem*.

## Utilização de instâncias

Para usar uma instância de uma classe, você precisa ter uma variável que aponte para o objeto. Uma variável ponteiro traz a localização de um objeto na memória, e não o objeto propriamente dito. (Ou seja, “aponta para” o objeto.) Uma variável que aponte para um objeto **Party** é declarada da seguinte forma:

```
Party *partyInstance;
```

A criação desse ponteiro não cria um objeto **Party** – apenas uma variável que possa apontar para um objeto **Party**.

Essa variável é chamada de `partyInstance`. Observe que o nome dessa variável não começa com um sublinhado; ela não é uma variável de instância. Ela serve para apontar para uma instância da classe **Party**.

## Criação de objetos

Um objeto tem um tempo de vida: ele é criado, envia mensagens, e depois é destruído quando deixa de ser necessário.

Para criar um objeto, você envia uma mensagem `alloc` para uma classe. Em resposta, a classe cria um objeto na memória (no heap, assim como `malloc()` criaria) e lhe dá o endereço do objeto, o qual você armazena em uma variável:

```
Party *partyInstance = [Party alloc];
```

Você cria um ponteiro para uma instância para que você possa enviar mensagens a ela. A primeira mensagem que *sempre* deve ser enviada a uma instância recém-alocada é uma mensagem de inicialização. Mesmo que o envio da mensagem `alloc` para uma classe crie uma instância, essa instância não estará pronta para funcionar até que seja inicializada.

```
Party *partyInstance = [Party alloc];
[partyInstance init];
```

Como um objeto deve ser alocado e inicializado antes de poder ser usado, você sempre combina essas duas mensagens em uma linha.

```
Party *partyInstance = [[Party alloc] init];
```

Quando duas mensagens são combinadas em uma única linha de código, isso se chama *envio de mensagem aninhada*. Os parênteses mais internos são avaliados primeiro, então a mensagem `alloc` é a primeira a ser enviada à classe **Party**. Isso retorna um ponteiro para uma instância nova e não inicializada de **Party** para a qual é enviada a mensagem `init`. Isso retorna um ponteiro para a instância inicializada que você armazenou em sua variável de ponteiro.

## Envio de mensagens

O que fazer com uma instância que foi inicializada? Você envia a ela mais mensagens.

Vamos analisar mais de perto a anatomia da mensagem. Uma mensagem está sempre contida entre colchetes. Dentro de um par de colchetes, uma mensagem tem três partes:

|                     |                                                                       |
|---------------------|-----------------------------------------------------------------------|
| <i>destinatário</i> | um ponteiro para o objeto ao qual se solicita a execução de um método |
| <i>seletor</i>      | nome do método a ser executado                                        |
| <i>argumentos</i>   | valores fornecidos como parâmetros para o método                      |

Por exemplo, uma festa pode ter uma lista de participantes à qual você pode acrescentar nomes enviando à festa a mensagem **addAttendee:**:

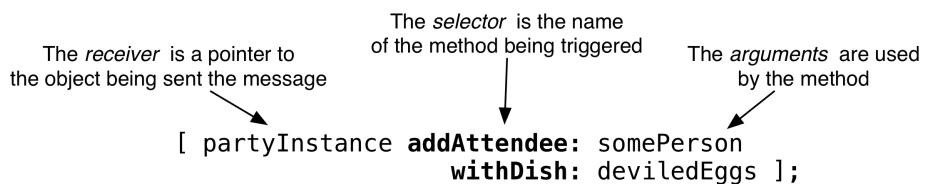
```
[partyInstance addAttendee:somePerson];
```

O envio da mensagem **addAttendee:** para **partyInstance** (destinatário) executa o método **addAttendee:** (identificado pelo seletor) e passa **somePerson** (um argumento).

A mensagem **addAttendee:** tem um argumento. Os métodos do Objective-C suportam vários argumentos ou nenhum. A mensagem **init**, por exemplo, não tem nenhum argumento.

Um participante da festa pode precisar confirmar a presença e informar ao anfitrião qual petisco vai trazer. Portanto, a classe **Party** pode ter outro método chamado **addAttendee:withDish:**. Essa mensagem suporta dois argumentos: o participante e o petisco. Cada argumento é pareado com um rótulo no seletor, e cada rótulo termina com um ponto e vírgula. O seletor é o agrupamento de todos os rótulos (Figure 2.2).

Figure 2.2 Partes do envio de uma mensagem



O pareamento de rótulos e argumentos é uma característica importante do Objective-C. Em outras linguagens, esse método teria a seguinte forma:

```
partyInstance.addAttendeeWithDish(somePerson, deviledEggs);
```

Nestas linguagens, não fica totalmente óbvio do que se trata cada um dos argumentos enviados a essa função. No Objective-C, entretanto, cada argumento é pareado com o rótulo apropriado.

```
[partyInstance addAttendee:somePerson withDish:deviledEggs];
```

É preciso se acostumar, mas eventualmente, os programadores de Objective-C passam a apreciar a clareza dos argumentos interpostos no seletor. O segredo é se lembrar de que, para cada par de colchetes, apenas uma mensagem está sendo enviada. Embora **addAttendee:withDish:** tenha dois rótulos, continua sendo apenas uma mensagem, e o envio dessa mensagem resulta na execução de apenas um método.

No Objective-C, o nome do método é o que o torna único. Portanto, uma classe não pode ter dois métodos com o mesmo nome. Dois nomes de métodos podem ter o mesmo rótulo individual, contanto que o nome de cada método como um todo seja diferente. Por exemplo, nossa classe **Party** tem dois métodos, **addAttendee:** e **addAttendee:withDish:**. Tratam-se de dois métodos diferentes, que não compartilham nenhum código.

Além disso, observe a distinção entre uma *mensagem* e um *método*: o método é um pedaço de código que pode ser executado, e uma mensagem é o ato de pedir que uma classe ou objeto execute um método. O nome de uma mensagem sempre corresponde ao nome do método que deve ser executado.

## Destrução de objetos

Para destruir um objeto, você precisa definir a variável que aponta para ele para **nil**.

```
partyInstance = nil;
```

Essa linha de código destrói o objeto para o qual a variável **partyInstance** está apontando e define o valor da variável **partyInstance** para **nil**. (Na verdade, é um pouquinho mais complicado do que isso, e você aprenderá sobre gerenciamento de memória no próximo capítulo.)

O valor **nil** é o ponteiro nulo. (Os programadores de C conhecem como **NULL**. Os programadores de Java conhecem como **null**.) Um ponteiro com valor **nil** é tipicamente usado para representar a ausência de um objeto. Por exemplo, a festa poderia ter um local. Enquanto o organizador da festa ainda está determinando onde ela será realizada, **venue** aponta para **nil**. Isso permite que você faça o seguinte:

```
if (venue == nil) {
    [organizer remindToFindVenueForParty];
}
```

Programadores de Objective-C geralmente usam a forma abreviada para determinar quando um ponteiro é `nil`:

```
if (!venue) {
    [organizer remindToFindVenueForParty];
}
```

Como o operador `!` significa “não,” ele seria lido como “se não houver um local” e avaliado como verdadeiro se `venue` fosse `nil`.

Se você enviar uma mensagem a uma variável definida como `nil`, nada acontece. Em outras linguagens, é ilegal enviar uma mensagem a um ponteiro nulo, então frequentemente se vê o seguinte:

```
// Is venue non-nil?
if (venue) {
    [venue sendConfirmation];
}
```

No Objective-C, essa verificação é desnecessária, já que a mensagem enviada a `nil` é ignorada. Portanto, você pode simplesmente enviar uma mensagem sem a verificação de `nil`:

```
[venue sendConfirmation];
```

Se o local ainda não foi escolhido, você não enviará a confirmação a lugar nenhum. (Uma dica: se o seu programa não está fazendo nada e deveria estar fazendo alguma coisa, o culpado quase sempre é um ponteiro `nil` inesperado.)

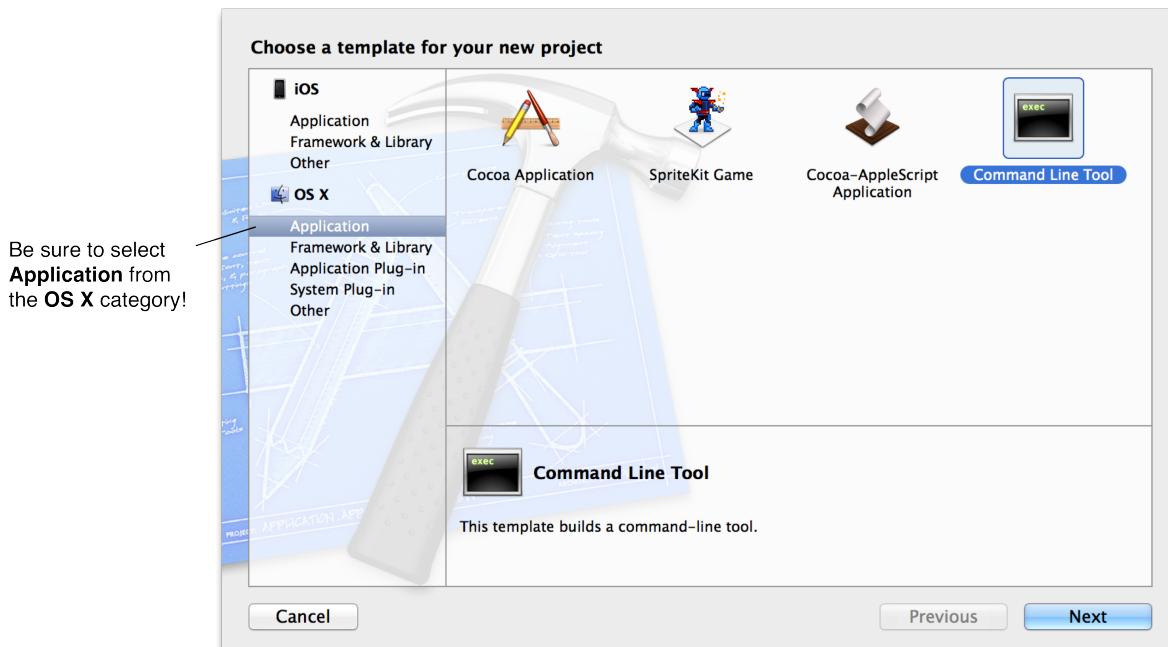
Chega de teoria. É hora de um pouco de prática e um novo projeto.

## Iniciando o RandomItems

Este novo projeto não é um aplicativo para iOS; é um programa de linha de comando. Criar um programa de linha de comando permitirá que o seu foco seja o Objective-C, sem a distração de uma interface de usuário. Ainda assim, os conceitos que você aprenderá aqui e no Chapter 3 são essenciais para o desenvolvimento de iOS. Você voltará a ler sobre a criação de aplicativos iOS no Chapter 4.

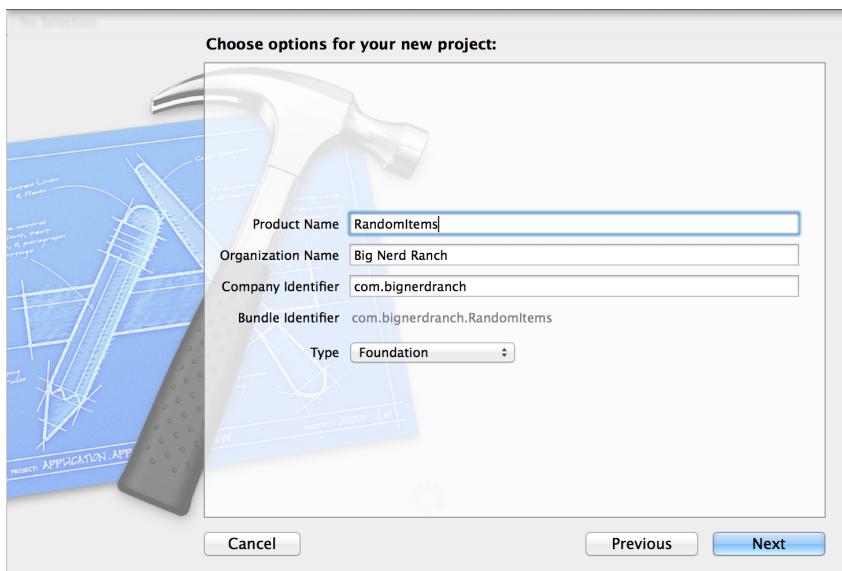
Abra o Xcode e selecione `File → New → Project....`. Na tabela da esquerda da seção OS X, clique em `Application` e selecione `Command Line Tool` no painel superior, conforme mostrado na Figure 2.3.

Figure 2.3 Criação de uma ferramenta de linha de comando



Clique no botão Next e, na folha que aparece, nomeie o produto RandomItems e selecione Foundation como tipo (Figure 2.4).

Figure 2.4 Nomeando o projeto



Clique em Next e você terá a opção de salvar o projeto. Salve-o em um local seguro – você reutilizará partes deste código em projetos futuros.

Na versão inicial de RandomItems, você criará um *array* de quatro strings. Um array é uma lista ordenada de ponteiros para outros objetos. Os ponteiros de um array são acessados por um índice. (Outras linguagens podem chamar de lista ou vetor.) Os arrays têm base zero; o primeiro item de um array está sempre no índice 0.

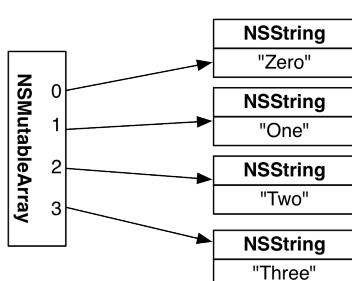
Depois que o array for criado, você executará um loop no array e exibirá cada string. O resultado será exibido no console do Xcode:

Figure 2.5 Saída do console

```
2013-12-08 18:37:03.481 RandomItems[4208:303] Zero
2013-12-08 18:37:03.483 RandomItems[4208:303] One
2013-12-08 18:37:03.484 RandomItems[4208:303] Two
2013-12-08 18:37:03.484 RandomItems[4208:303] Three
Program ended with exit code: 0
```

O seu programa precisará de cinco objetos: uma instância de **NSMutableArray** e quatro instâncias de **NSString** (Figure 2.6).

Figure 2.6 Instância de **NSMutableArray** contendo ponteiros para as instâncias de **NSString**

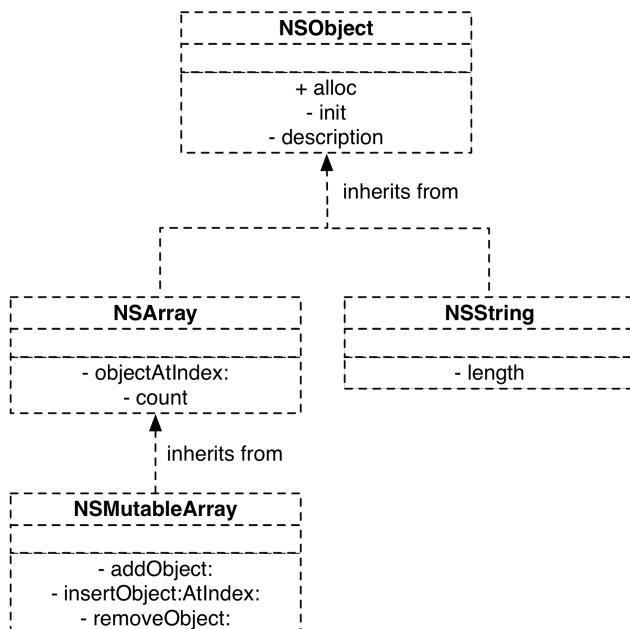


No Objective-C, um array não contém os objetos que a ele pertencem; em vez disso, contém um ponteiro para cada objeto. Quando um objeto é adicionado a um array, o endereço desse objeto na memória é armazenado dentro do array.

Agora, vamos considerar as classes **NSMutableArray** e **NSString**. **NSMutableArray** é uma *subclasse* de **NSArray**. As classes têm uma hierarquia, e todas as classes têm exatamente uma superclasse – exceto a classe raiz da hierarquia como um todo: **NSObject**. Uma classe herda o comportamento da sua superclasse.

A Figure 2.7 mostra a hierarquia de classes para **NSMutableArray** e **NSString** junto com alguns métodos implementados por cada classe.

Figure 2.7 Hierarquia de classes



Como superclasse principal, o papel da **NSObject** é implementar o comportamento básico de cada um dos objetos do Cocoa Touch. Toda classe herda os métodos e as variáveis de instância definidos na **NSObject**. Dois métodos que a **NSObject** implementa são **alloc** e **init** (Figure 2.7). Portanto, esses métodos podem ser usados para criar uma instância de qualquer classe.

Uma subclasse adiciona métodos e variáveis de instância para expandir o comportamento da sua superclasse:

- **NSString** adiciona o comportamento para armazenar e manusear strings, incluindo o método **length** que retorna o número de caracteres em uma string.
- **NSArray** adiciona o comportamento para listas ordenadas, incluindo o acesso a um objeto em um dado índice (**objectAtIndex:**) e a obtenção do número de objetos em um array (**count**).
- **NSMutableArray** amplia as capacidades da classe **NSArray** adicionando as capacidades para adicionar e remover ponteiros.

## Criação e preenchimento de um array

Vamos usar essas classes em um código real. No navegador de projetos, encontre e selecione o arquivo chamado `main.m`. Ao abri-lo, você verá que algum código foi escrito para você – em especial, uma função `main`, que é o ponto de entrada de qualquer aplicativo em C ou Objective-C.

No `main.m`, exclua a linha de código que chama **NSLogs** para exibir “Hello, World!” e troque por um código que crie uma instância de **NSMutableArray**, adicione um total de quatro objetos ao array mutável e, depois, destrua o array.

```

#import <Foundation/Foundation.h>

int main (int argc, const char * argv[])
{
    @autoreleasepool {

        // insert code here...
        NSLog(@"Hello, World!");

        // Create a mutable array object, store its address in items variable
        NSMutableArray *items = [[NSMutableArray alloc] init];

        // Send the message addObject: to the NSMutableArray pointed to
        // by the variable items, passing a string each time
        [items addObject:@"One"];
        [items addObject:@"Two"];
        [items addObject:@"Three"];

        // Send another message, insertObjectAtIndex:, to that same array object
        [items insertObject:@"Zero" atIndex:0];

        // Destroy the mutable array object
        items = nil;
    }
    return 0;
}

```

Os objetos que você adicionou e inseriu no array são instâncias de **NSString**. Você pode criar uma instância de **NSString** prefixando uma string de caracteres com o símbolo @:

```
NSString *myString = @"Hello, World!";
```

## Iteração em um array

Agora que você tem um array `items` com strings, você vai fazer a iteração no array, acessar cada string e enviar o resultado para o console.

Você poderia escrever essa operação como um loop `for`:

```

for (int i = 0; i < [items count]; i++) {
    NSString *item = [items objectAtIndex:i];
    NSLog(@"%@", item);
}

```

Como arrays têm base zero, o contador começa em 0 e para em um a menos do que o resultado do envio da mensagem `count` ao array de itens. Dentro do loop, você envia a mensagem `objectAtIndex:` para recuperar o objeto no índice atual antes de imprimi-lo.

A informação retornada do `count` é importante porque, se você pedir um objeto de um array com um índice igual ou maior que o número de objetos do array, ocorrerá uma exceção. (Em algumas linguagens, exceções ocorrem e são detectadas o tempo todo. No Objective-C, exceções são consideradas erros do programador e devem ser corrigidas no código em vez de serem tratadas no tempo de execução. Falaremos mais sobre exceções no fim deste capítulo.)

Este código funcionaria bem, mas o Objective-C oferece uma opção melhor para fazer a iteração em um array, chamada de *enumeração rápida*. A enumeração rápida apresenta uma sintaxe menor que um loop `for` tradicional e muito menos suscetível a erros. Em alguns casos, ela será mais rápida.

No `main.m`, adicione o código seguinte, que utiliza a enumeração rápida para iterar no array `items`.

```

int main (int argc, const char * argv[])
{
    @autoreleasepool {

        // Create a mutable array object, store its address in items variable
        NSMutableArray *items = [[NSMutableArray alloc] init];

        // Send the message addObject: to the NSMutableArray pointed to
        // by the variable items, passing a string each time
        [items addObject:@"One"];
        [items addObject:@"Two"];
        [items addObject:@"Three"];

        // Send another message, insertObjectAtIndex:, to that same array object
        [items insertObject:@"Zero" atIndex:0];

        // For every item in the items array ...
        for (NSString *item in items) {
            // Log the description of item
            NSLog(@"%@", item);
        }

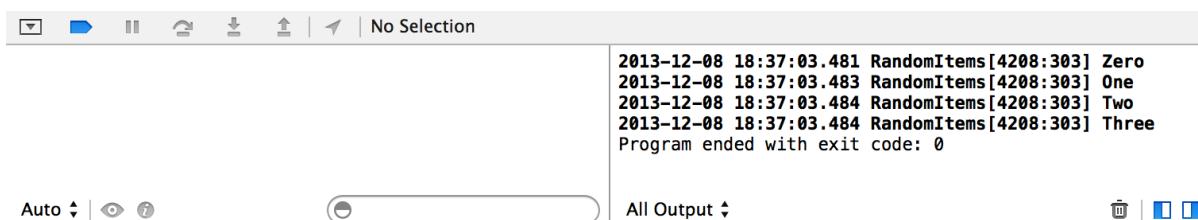
        items = nil;
    }
    return 0;
}

```

A enumeração rápida tem uma limitação: você não pode utilizá-la caso precise adicionar ou remover objetos no loop. Tentar fazê-lo resultará na ocorrência de uma exceção. Se você precisar adicionar ou remover objetos, deve usar um loop `for` normal com uma variável de contador.

Compile e execute o aplicativo (Command-R). Um novo painel será exibido na parte inferior da janela do Xcode. Esta é a *área de depuração*. No lado direito dessa área encontra-se o *console* e seu resultado.

Figure 2.8 Resultado no console



Se precisar, você pode redimensionar a área de depuração e seus painéis, arrastando os respectivos frames. (Na verdade, você pode redimensionar qualquer parte da janela da área de trabalho dessa maneira.)

Você concluiu a primeira fase do programa `RandomItems`. Antes de passar para a próxima fase, vamos examinar a função `NSLog` e strings de formatação.

## Strings de formatação

`NSLog()` coleta um número variável de argumentos e exibe uma string no registro. No Xcode, o registro é exibido no console. O primeiro argumento da função `NSLog()` é obrigatório. Ele é uma instância de `NSString` e é chamado de *string de formatação*.

Uma string de formatação contém texto e um número de *tokens*. Cada token (também chamado de especificação de formato) vem prefixado com um símbolo de porcentagem (%). Cada argumento adicional passado à função substitui um token na string de formatação.

Os tokens especificam o tipo de argumento ao qual correspondem. Veja um exemplo:

```

int a = 1;
float b = 2.5;
char c = 'A';
NSLog(@"Integer: %d Float: %f Char: %c", a, b, c);

```

O resultado seria

```
Integer: 1 Float: 2.5 Char: A
```

As strings de formatação do Objective-C funcionam da mesma maneira que em C. Contudo, o Objective-C adiciona mais um token: %@. Esse token corresponde a um argumento cujo tipo é “um ponteiro para qualquer objeto”.

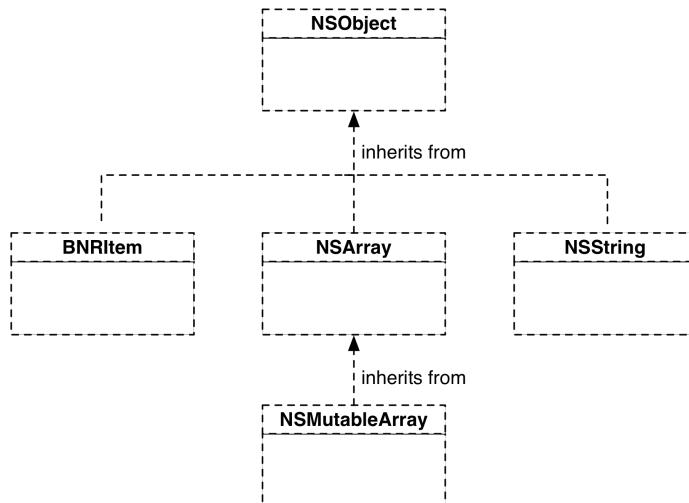
Quando %@ está presente na string de formatação, em vez de o token ser substituído pelo argumento correspondente, a mensagem **description** é enviada a esse argumento. O método **description** retorna uma instância de **NSString** que substitui o token.

Como uma mensagem é enviada ao argumento correspondente ao token %@, esse argumento precisa ser um objeto. Na Figure 2.7, você pode notar que **description** é um método na classe **NSObject**. Portanto, toda classe implementa **description**, por isso você pode usar o token %@ com qualquer objeto.

## Criação de subclasses em uma classe do Objective-C

Nesta seção, você vai criar uma nova classe chamada **BNRItem**. **BNRItem** será uma subclass de **NSObject**.

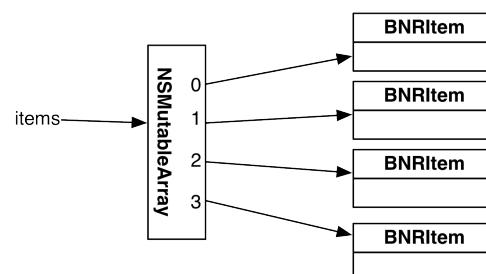
Figure 2.9 Hierarquia de classes incluindo **BNRItem**



Uma instância de **BNRItem** representará algo que uma pessoa possui no mundo real, como um laptop, uma bicicleta ou uma mochila. Em termos de Modelo-Visão-Controlador, **BNRItem** é uma classe de modelo. Uma instância de **BNRItem** armazena informações sobre um bem.

Após criar a classe **BNRItem**, você preencherá o array **items** com instâncias de **BNRItem** em vez de **NSString**.

Figure 2.10 Uma classe diferente de itens

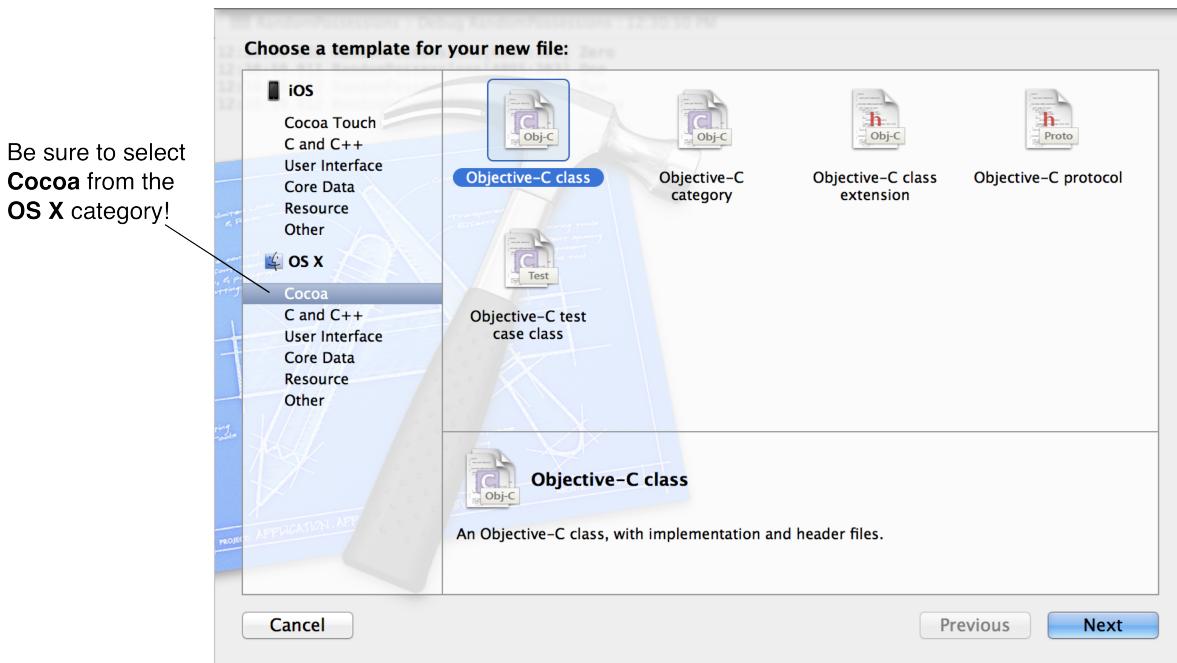


Mais adiante no livro, você reutilizará **BNRItem** em um aplicativo para iOS complexo.

## Criação de uma subclasse NSObject

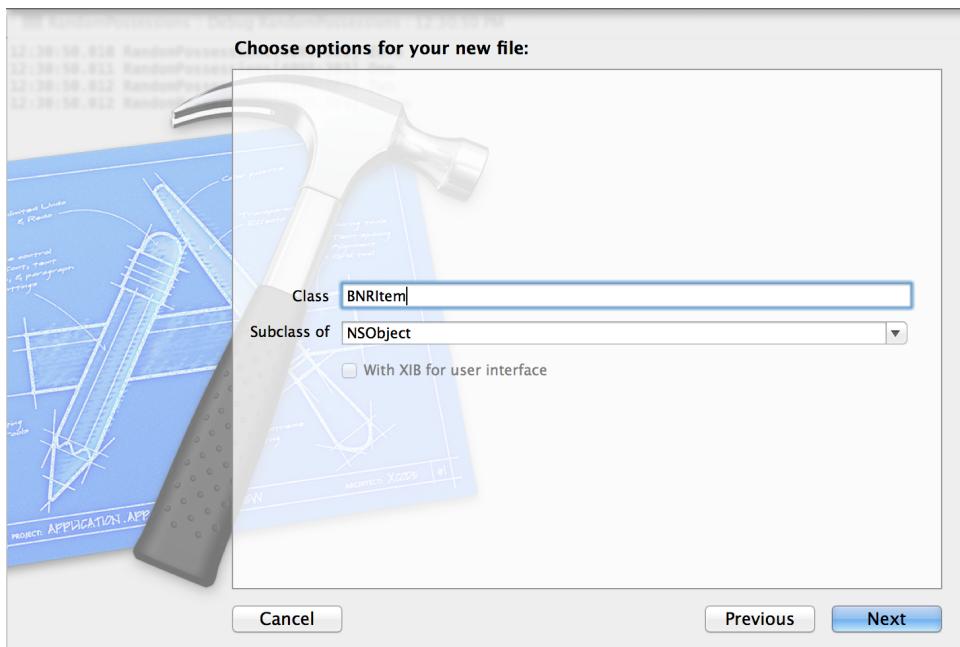
Para criar uma nova classe no Xcode, selecione File → New → File.... Na tabela à esquerda do painel exibido, selecione Cocoa na seção OS X. Depois, selecione Objective-C class no painel superior e clique em Next (Figure 2.11).

Figure 2.11 Criação de classes



No próximo painel, nomeie esta nova classe **BNRItem** e insira **NSObject** como sua superclasse, conforme mostrado na Figure 2.12.

Figure 2.12 Seleção de superclasse



Clique em Next e você terá a opção de selecionar onde salvar os arquivos para essa classe. Pode ser a localização padrão. Certifique-se de que a caixa do destino RandomItems esteja selecionada. Clique em Create.

No navegador de projetos, encontre os arquivos de classe para **BNRItem** # **BNRItem.h** e **BNRItem.m**:

- **BNRItem.h** é o *arquivo de cabeçalho* (conhecido também como *arquivo de interface*). Esse arquivo declara o nome da nova classe, a superclasse correspondente, as variáveis de instância de cada uma das instâncias dessa classe e todos os métodos implementados por essa classe.
- **BNRItem.m** é o arquivo de implementação, que contém o código dos métodos implementados pela classe.

Todas as classes do Objective-C têm esses dois arquivos. O arquivo de cabeçalho é como um manual do usuário para uma instância de uma classe, e o arquivo de implementação corresponde aos detalhes de engenharia que definem como ela realmente funciona.

Selecione o **BNRItem.h** no navegador de projetos. O conteúdo do arquivo é semelhante ao seguinte:

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

@interface BNRItem : NSObject

@end
```

Para declarar uma classe no Objective-C, use a palavra-chave `@interface` seguida pelo nome da nova classe. Após o dois pontos vem o nome da superclasse. O Objective-C permite apenas herança simples, portanto, cada classe pode ter apenas uma superclasse:

```
@interface ClassName : SuperclassName
```

A diretiva `@end` indica que a declaração da classe chegou ao fim.

Observe os prefixos `@`. O Objective-C conserva as palavras-chave da linguagem C. Outras palavras-chave específicas do Objective-C são diferenciadas pelo prefixo `@`.

## Variáveis de instância

Um “item”, em nosso mundo, vai ter um nome, um número de série, um valor e uma data de criação. Essas serão as variáveis de instância de **BNRItem**.

As variáveis de instância de uma classe são declaradas entre chaves, imediatamente depois da declaração da classe.

No **BNRItem.h**, acrescente um conjunto de chaves e quatro variáveis de instância para a classe **BNRItem**:

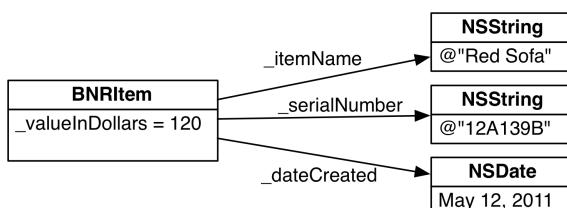
```
#import <Foundation/Foundation.h>

@interface BNRItem : NSObject
{
    NSString *_itemName;
    NSString *_serialNumber;
    int _valueInDollars;
    NSDate *_dateCreated;
}

@end
```

Agora, toda instância de **BNRItem** terá um lugar para um inteiro simples e três lugares para armazenar ponteiros para os objetos, especificamente duas instâncias de **NSString** e uma instância de **NSDate**. (Lembre-se, o `*` indica que a variável é um ponteiro.) A Figure 2.13 mostra um exemplo de uma instância de **BNRItem** após a atribuição de valores a suas variáveis de instância.

Figure 2.13 Uma instância de **BNRItem**



Observe que a Figure 2.13 mostra quatro objetos no total: a instância de **BNRItem**, duas instâncias de **NSString** e uma instância de **NSDate**. Cada objeto existe independentemente e fora dos outros. As variáveis de instância do objeto **BNRItem** são ponteiros para os outros objetos, não para os objetos propriamente ditos.

Por exemplo, cada instância de **BNRItem** tem uma variável de instância ponteiro chamada `_itemName`. A `_itemName` do objeto **BNRItem**, conforme mostrado na Figure 2.13, aponta para um objeto **NSString** com o conteúdo "Red Sofa". A string "Red Sofa" não reside dentro do objeto **BNRItem**. O objeto **BNRItem** sabe onde a string "Red Sofa" reside na memória e armazena o endereço correspondente como `_itemName`. Uma forma de pensar nesse relacionamento é: "o objeto **BNRItem** chama essa string como sua `_itemName`."

A situação é diferente para a variável de instância `_valueInDollars`. Essa variável de instância *não* é um ponteiro para outro objeto; é apenas um `int`. O `int` propriamente dito reside no objeto **BNRItem**.

A ideia dos ponteiros pode não ser fácil de entender no começo. No próximo capítulo, você aprenderá mais sobre objetos, ponteiros e variáveis de instância, e, ao longo deste livro, você verá diagramas de objetos como a Figure 2.13 para elucidar a diferença entre um objeto e um ponteiro para um objeto.

## Acesso a variáveis de instância

Agora que as instâncias de **BNRItem** têm variáveis de instância, você precisa de uma forma de acessá-las e modificar seus valores. Em linguagens orientadas a objetos, chamamos os métodos que obtêm e modificam as variáveis de instância de *acessores*. Eles são conhecidos individualmente como *getters* e *setters*. Sem esses métodos, um objeto não consegue acessar as variáveis de instância de outro objeto.

No **BNRItem.h**, declare os métodos acessores para as variáveis de instância da classe **BNRItem**. Você precisa de getters e setters para `_valueInDollars`, `_itemName` e `_serialNumber`. A variável de instância `_dateCreated` será somente para leitura, portanto, ela precisa apenas de um método getter.

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

@interface BNRItem : NSObject
{
    NSString *_itemName;
    NSString *_serialNumber;
    int _valueInDollars;
    NSDate *_dateCreated;
}

- (void)setItemName:(NSString *)str;
- (NSString *)itemName;

- (void)setSerialNumber:(NSString *)str;
- (NSString *)serialNumber;

- (void)setValueInDollars:(int)v;
- (int)valueInDollars;

- (NSDate *)dateCreated;
@end
```

No Objective-C, o nome de um método setter é **set** mais o nome em letras maiúsculas da variável de instância que ele está modificando – nesse caso, **setItemName:**. Em outras linguagens, o nome do método getter provavelmente seria **getItemName**. Entretanto, no Objective-C, o nome do método getter é apenas o nome da variável de instância. Algumas das partes mais legais da biblioteca Cocoa Touch pressupõem que as suas classes seguirão essa convenção; portanto, os programadores mais descolados do Cocoa Touch fazem sempre dessa forma.

(Para aqueles que já têm experiência em Objective-C, falaremos sobre propriedades no próximo capítulo.)

Depois, abra o arquivo de implementação de **BNRItem**, **BNRItem.m**.

No topo de qualquer arquivo de implementação, sempre é importado o arquivo de cabeçalho daquela classe. A implementação de uma classe precisa saber como ela foi declarada. (Importar um arquivo é o mesmo que incluir um arquivo na linguagem C, exceto que há a garantia de que o arquivo será incluído apenas uma vez.)

Depois da instrução de importação vem o bloco de implementação, que começa com a palavra-chave `@implementation`, seguida pelo nome da classe que está sendo implementada. Todas as definições de métodos

no arquivo de implementação ficam dentro deste bloco de implementação. Os métodos são definidos até que o bloco seja encerrado com a palavra-chave `@end`.

No `BNRItem.m`, exclua tudo que o template tenha possivelmente adicionado entre `@implementation` e `@end`. Em seguida, defina os métodos acessores para as variáveis de instância que você declarou no `BNRItem.h`.

```
#import "BNRItem.h"

@implementation BNRItem

- (void)setItemName:(NSString *)str
{
    _itemName = str;
}
- (NSString *)itemName
{
    return _itemName;
}

- (void)setSerialNumber:(NSString *)str
{
    _serialNumber = str;
}

- (NSString *)serialNumber
{
    return _serialNumber;
}

- (void)setValueInDollars:(int)v
{
    _valueInDollars = v;
}

- (int)valueInDollars
{
    return _valueInDollars;
}

- (NSDate *)dateCreated
{
    return _dateCreated;
}

@end
```

Observe que cada método setter define a variável de instância para qualquer coisa que seja passada como um argumento, e cada método getter retorna o valor da variável de instância.

Nesse ponto, verifique e corrija qualquer erro no seu código sobre o qual o Xcode está avisando. Alguns possíveis culpados são erros de digitação e falta de ponto e vírgula.

Vamos testar a sua nova classe e os respectivos métodos acessores. No `main.m`, primeiro importe o arquivo de cabeçalho para a classe `BNRItem`.

```
#import <Foundation/Foundation.h>
#import "BNRItem.h"

int main (int argc, const char * argv[])
{
    ...
```

Por que você importa o cabeçalho de classe `BNRItem.h`, mas não importa, digamos, o `NSMutableArray.h`? A classe `NSMutableArray` é proveniente do framework Foundation, assim ela é incluída quando você importa o `Foundation/Foundation.h`. Por outro lado, a classe `BNRItem` existe em seu próprio arquivo, então você precisa importá-la explicitamente para `main.m`. Se não fizer isso, o compilador não sabe que ela existe e reclama bastante.

Em seguida, crie uma instância de `BNRItem` e registre suas variáveis de instância no console.

```

int main (int argc, const char * argv[])
{
    @autoreleasepool {

        NSMutableArray *items = [[NSMutableArray alloc] init];
        [items addObject:@"One"];
        [items addObject:@"Two"];
        [items addObject:@"Three"];
        [items insertObject:@"Zero" atIndex:0];

        // For every item in the array pointed to by items...
        for (NSString *item in items) {
            // print a description
            NSLog(@"%@", item);
        }

        BNRItem *item = [[BNRItem alloc] init];
        NSLog(@"%@", [item itemName], [item dateCreated],
                 [item serialNumber], [item valueInDollars]);

        items = nil;
    }

    return 0;
}

```

Compile e execute o aplicativo. No final da saída, você verá três strings (`null`) e um 0. Esses são os valores das variáveis de instância da sua nova instância de `BNRItem`.

Figure 2.14 Valores de variáveis de instância no console

```

2013-12-08 18:43:11.237 RandomItems[4239:303] Zero
2013-12-08 18:43:11.239 RandomItems[4239:303] One
2013-12-08 18:43:11.239 RandomItems[4239:303] Two
2013-12-08 18:43:11.240 RandomItems[4239:303] Three
2013-12-08 18:43:11.240 RandomItems[4239:303] (null) (null) (null) 0
Program ended with exit code: 0

```

Quando um objeto é criado, todas as suas variáveis de instância são “zeradas”. O ponteiro para um objeto aponta para `nil`; um primitivo como `int` tem o valor de 0.

Para dar às variáveis de instância do objeto `BNRItem` valores mais interessantes, você precisa criar novos objetos e passá-los como argumentos aos métodos setter.

No `main.m`, insira o seguinte código:

```

// Notice we are omitting some of the surrounding code
...

BNRItem *item = [[BNRItem alloc] init];

// This creates an NSString, "Red Sofa" and gives it to the BNRItem
[item setName:@"Red Sofa"];

// This creates an NSString, "A1B2C" and gives it to the BNRItem
[item setSerialNumber:@"A1B2C"];

// This sends the value 100 to be used as the valueInDollars of this BNRItem
[item setValueInDollars:100];

NSLog(@"%@", [item itemName], [item dateCreated],
         [item serialNumber], [item valueInDollars]);

...

```

Compile e execute o aplicativo. Você verá os valores das três variáveis de instância. Você verá ainda (`null`) para `dateCreated`. Mais adiante no capítulo, você cuidará da atribuição de um valor a essa variável de instância quando um objeto for criado.

Figure 2.15 Mais valores interessantes

```
2013-12-08 18:44:56.551 RandomItems[4254:303] Zero
2013-12-08 18:44:56.553 RandomItems[4254:303] One
2013-12-08 18:44:56.553 RandomItems[4254:303] Two
2013-12-08 18:44:56.554 RandomItems[4254:303] Three
2013-12-08 18:44:56.554 RandomItems[4254:303] Red Sofa (null) A1B2C 100
Program ended with exit code: 0
```

## Utilização de sintaxe de ponto

Para obter e modificar uma variável de instância, você pode enviar mensagens de acessores explícitos:

```
BNRItem *item = [[BNRItem alloc] init];
// set valueInDollars by sending an explicit message
[item setValueInDollars:5];
// get valueInDollars by sending an explicit message
int value = [item valueInDollars];
```

Ou você pode utilizar a *sintaxe de ponto*, também conhecida como *notação de ponto*. Este é o mesmo código usando a sintaxe de ponto:

```
BNRItem *item = [[BNRItem alloc] init];
// set valueInDollars using dot syntax
item.valueInDollars = 5;
// get valueInDollars using dot syntax
int value = item.valueInDollars;
```

O destinatário (`item`) é seguido por um `.`, seguido pelo nome da variável de instância sem o sublinhado à esquerda (`valueInDollars`).

Observe que a sintaxe é a mesma tanto para definir quanto para obter a variável de instância (`item.valueInDollars`); a diferença é em que lado do operador de distribuição ela está.

Não há diferença no tempo de execução entre mensagens de acessor e sintaxe de ponto; o código compilado é o mesmo e qualquer uma das sintaxes chamarão os métodos `valueInDollars` e `setValueInDollars:` que você acabou de implementar.

Hoje em dia, os modernos programadores em Objective-C tendem a usar a sintaxe de ponto para chamar acessores. Ela facilita a leitura do código, principalmente quando, tradicionalmente, haveria chamadas de mensagem aninhada. E também é consistente com o código da Apple. É o que faremos neste livro.

No `main.m`, atualize o código para usar a sintaxe de ponto para definir as variáveis de instância e para obtê-las como parte da string de formatação.

```
...
BNRItem *item = [[BNRItem alloc] init];
// This creates an NSString, "Red Sofa" and gives it to the BNRItem
[item setName:@"Red Sofa"];
item.itemName = @"Red Sofa";
// This creates an NSString, "A1B2C" and gives it to the BNRItem
[item setSerialNumber:@"A1B2C"];
item.serialNumber = @"A1B2C";
// This sends the value 100 to be used as the valueInDollars of this BNRItem
[item setValueInDollars:100];
item.valueInDollars = 100;
NSLog(@"%@", [item itemName], [item dateCreated],
        [item serialNumber], [item valueInDollars]);
NSLog(@"%@", item.itemName, item.dateCreated,
        item.serialNumber, item.valueInDollars);
...

```

## Métodos de classe vs. de instância

Os métodos podem ser de dois tipos: *métodos de instância* e *métodos de classe*. O método de classe cria novas instâncias da classe, ou recupera alguma propriedade global da classe. O método de instância opera em uma instância específica da classe. Por exemplo, os acessores que você acabou de implementar são todos métodos de instância. Eles são utilizados para definir ou obter as variáveis de instância de um objeto em particular.

Para chamar um método de instância, você envia a mensagem a uma instância da classe. Para chamar um método de classe, você envia a mensagem para a própria classe.

Por exemplo, quando você criou uma instância de **BNRItem**, você enviou **alloc** (um método de classe) para a classe **BNRItem** e, em seguida, **init** (um método de instância) para a instância de **BNRItem** retornada de **alloc**.

O método **description** é um método de instância. Na próxima seção, você vai implementar o método **description** em **BNRItem** para retornar um objeto **NSString** que descreva uma instância de **BNRItem**. Mais adiante no capítulo, você implementará um método de classe para criar uma instância de **BNRItem** usando valores aleatórios.

## Sobrescrevendo métodos

Uma subclasse também pode sobrescrever métodos de sua superclasse. Por exemplo, o envio de uma mensagem **description** a uma instância de **NSObject** retorna a classe do objeto e seu endereço na memória como uma instância de **NSString**, conforme segue:

```
<BNRQuizViewController: 0x4b222a0>
```

Uma subclasse de **NSObject** pode sobrescrever esse método e retornar um objeto **NSString** que descreva melhor uma instância dessa subclasse. Por exemplo, a classe **NSString** sobrescreve **description** para retornar a própria string. A classe **NSArray** sobrescreve **description** para retornar a descrição de cada objeto no array.

Como **BNRItem** é uma subclasse de **NSObject** (a classe que originalmente declara o método **description**), ao implementar novamente **description** na **BNRItem**, você está *sobrescrevendo-o*.

Para sobrescrever um método, você só precisa defini-lo no arquivo de implementação; você não precisa declará-lo no arquivo de cabeçalho, porque ele já foi declarado pela superclasse.

No **BNRItem.m**, sobrescreva o método **description**. O código para a implementação de um método pode estar em qualquer lugar entre **@implementation** e **@end**, contanto que não esteja dentro das chaves de um método existente.

```
- (NSString *)description
{
    NSString *descriptionString =
        [[NSString alloc] initWithFormat:@"%@ (%@): Worth $%d, recorded on %@",

            self.itemName,
            self.serialNumber,
            self.valueInDollars,
            self.dateCreated];

    return descriptionString;
}
```

Observe o que você não está fazendo aqui: não está passando as variáveis de instância por nome (p. ex., `_itemName`). Em vez disso, você está chamando acessores (usando sintaxe de ponto). É uma boa prática usar métodos acessores para acessar variáveis de instância mesmo dentro de uma classe. É possível que um método acessor possa alterar alguma coisa da variável de instância que você está tentando acessar, e você quer se certificar de que ele tenha a oportunidade de fazer o que precisa.

Agora, sempre que você enviar a mensagem **description** para uma instância de **BNRItem**, ela retornará uma instância de **NSString** que melhor descreve a instância.

No **main.m**, substitua a instrução que exibe as variáveis de instância individualmente pela instrução que depende da implementação do método **description** pela **BNRItem**.

```

...
item.valueInDollars = 100;
NSLog(@"%@", item.itemName, item.dateCreated,
        item.serialNumber, item.valueInDollars);

// The %@ token is replaced with the result of sending
// the description message to the corresponding argument
NSLog(@"%@", item);
items = nil;

```

Compile e execute o aplicativo, e verifique os resultados no console.

Figure 2.16 Descrição de uma instância de **BNRItem**

```

2013-12-08 18:49:05.934 RandomItems[4277:303] Zero
2013-12-08 18:49:05.935 RandomItems[4277:303] One
2013-12-08 18:49:05.936 RandomItems[4277:303] Two
2013-12-08 18:49:05.936 RandomItems[4277:303] Three
2013-12-08 18:49:05.937 RandomItems[4277:303] Red Sofa (A1B2C): Worth $100, recorded on (null)
Program ended with exit code: 0

```

E se quiser criar um método de instância totalmente novo, um que você não esteja sobrescrevendo da superclasse? Você declara o novo método no arquivo de cabeçalho e o define no arquivo de implementação. Vejamos como isso funciona criando dois novos métodos de instância para inicializar uma instância de **BNRItem**.

## Inicializadores

Neste momento, a classe **BNRItem** tem apenas uma maneira de inicializar uma instância – o método **init**, o qual ela herda da classe **NSObject**. Nesta seção, você vai escrever dois métodos de inicialização adicionais, ou *inicializadores*, para **BNRItem**.

No **BNRItem.h**, declare dois inicializadores.

```

NSDate *_dateCreated;
}

- (instancetype)initWithitemName:(NSString *)name
    valueInDollars:(int)value
    serialNumber:(NSString *)sNumber;

- (instancetype)initWithitemName:(NSString *)name;
- (void)setitemName:(NSString *)str;

```

(Está curioso a respeito de **instancetype**? Aguarde um pouco – logo chegaremos lá.)

Cada inicializador começa com a palavra **init**. Nomear os inicializadores dessa forma não torna esses métodos diferentes de outros métodos de instância; trata-se apenas de uma convenção de nomenclatura. Entretanto, a comunidade do Objective-C adora convenções de nomenclatura, e você deve seguir-las. (Ignorar as convenções de nomenclatura no Objective-C resulta em problemas que são piores do que você pode imaginar.)

Um inicializador recebe argumentos que o objeto pode utilizar para se inicializar. Geralmente, uma classe tem inicializadores múltiplos, pois instâncias podem ter necessidades de inicialização diferentes. Por exemplo, o primeiro inicializador que você declarou recebe três argumentos, os quais são utilizados para configurar o nome, o valor e o número de série do item. Portanto, você precisa de todas essas informações para inicializar uma instância com esse método. E se você sabe apenas o nome do item? Então, você pode usar o segundo inicializador.

## O inicializador designado

Para cada classe, independentemente de sua quantidade de métodos inicializadores, um método é escolhido como o *inicializador designado*. O inicializador designado assegura que todas as variáveis de instância de um

objeto sejam válidas. (“Válido”, nesse contexto, significa que “quando você envia mensagens a este objeto depois de inicializá-lo, você pode prever a saída e nada ruim acontecerá”.)

Geralmente, o inicializador designado tem parâmetros para as variáveis de instância mais importantes e mais usadas de um objeto. A classe **BNRItem** tem quatro variáveis de instância, mas apenas três delas são graváveis. Portanto, o inicializador designado de **BNRItem** deve aceitar três argumentos e fornecer um valor dentro de sua implementação para `_dateCreated`.

No `BNRItem.h`, adicione um comentário indicando o inicializador designado:

```
NSDate *_dateCreated;  
}  
  
// Designated initializer for BNRItem  
- (instancetype)initWithItemName:(NSString *)name  
    valueInDollars:(int)value  
    serialNumber:(NSString *)sNumber;  
  
- (instancetype)initWithItemName:(NSString *)name;  
  
- (void)setItemName:(NSString *)str;
```

## instancetype

O tipo de retorno para ambos os inicializadores é `instancetype`. Essa palavra-chave somente pode ser utilizada para tipos de retorno, e ela relaciona o tipo de retorno com o destinatário. Os métodos `init` são sempre declarados para retornar `instancetype`.

Por que não definir o tipo de retorno como `BNRItem *`? Isso causaria um problema se a classe **BNRItem** viesse a ter subclasses. A subclasse herdaria todos os métodos de **BNRItem**, incluindo o inicializador e o tipo de retorno. Se uma instância da subclasse recebesse essa mensagem do inicializador, o que seria retornado? Não um ponteiro para uma instância de **BNRItem**, mas um ponteiro para uma instância da subclasse. Você poderia pensar: “Sem problemas. Eu vou sobrescrever o inicializador na subclasse para alterar o tipo de retorno.” Mas no Objective-C, não é possível ter dois métodos com o mesmo seletor e diferentes tipos de retorno (ou argumentos). Ao especificar que um método de inicialização retorne “uma instância do objeto destinatário”, você nunca vai precisar se preocupar com o que acontece nessa situação.

## id

Antes da palavra-chave `instancetype` ser introduzida no Objective-C, os inicializadores retornavam `id` (pronuncia-se “ai-di”). Esse tipo é definido como um “ponteiro para qualquer objeto”. (`id` é muito parecido com `void *` em C.). Até o momento em que este livro foi escrito, os templates de classe do Xcode ainda utilizam `id` como o tipo de retorno de inicializadores que foram adicionados no código padronizado. Imaginamos que isso mudará em breve.

Diferente do `instancetype`, o `id` pode ser usado como mais do que apenas um tipo de retorno. Você pode declarar variáveis ou parâmetros de método do tipo `id` quando não tiver certeza para que tipo de objeto a variável acabará apontando.

```
id objectOfUnknownType;
```

Você pode usar `id` quando utilizar a enumeração rápida para iterar em um array de tipos de objetos múltiplos ou desconhecidos:

```
for (id item in items) {  
    NSLog(@"%@", item);  
}
```

Observe que, como o `id` é definido como um “ponteiro para qualquer objeto”, você não inclui um `*` ao declarar uma variável ou parâmetro de método desse tipo.

## Implementação do inicializador designado

No `BNRItem.m`, implemente o inicializador designado dentro do bloco de implementação.

```

@implementation BNRItem

- (instancetype)initWithItemName:(NSString *)name
                           valueInDollars:(int)value
                            serialNumber:(NSString *)sNumber
{
    // Call the superclass's designated initializer
    self = [super init];

    // Did the superclass's designated initializer succeed?
    if (self) {
        // Give the instance variables initial values
        _itemName = name;
        _serialNumber = sNumber;
        _valueInDollars = value;
        // Set _dateCreated to the current date and time
        _dateCreated = [[NSDate alloc] init];
    }

    // Return the address of the newly initialized object
    return self;
}

```

Há muito sobre o que falar nesse código. Primeiro, observe que você definiu a variável de instância `_dateCreated` para apontar para uma nova instância de `NSDate`, que representa a data e a hora atuais.

Em seguida, analise a primeira linha do código nesta implementação. No inicializador designado, a primeira coisa que você deve fazer sempre é chamar o inicializador designado da superclasse usando `super`. A última coisa a fazer é retornar um ponteiro para o objeto que foi inicializado com êxito, usando `self`. Para entender o que acontece em um inicializador, você precisa saber mais a respeito de `self` e `super`.

## self

Dentro de um método, `self` é uma variável local implícita. Não é necessário declará-la, e ela é automaticamente definida para apontar para o objeto ao qual a mensagem foi enviada. (A maioria das linguagens orientadas a objetos tem esse conceito, mas algumas chamam de `this` em vez de `self`.) Geralmente, `self` é usado para que um objeto possa enviar uma mensagem a ele mesmo:

```

- (void)chickenDance
{
    [self pretendHandsAreBeaks];
    [self flapWings];
    [self shakeTailFeathers];
}

```

Na última linha de um método `init`, você sempre retorna o objeto recém-inicializado, para que o chamador possa atribuí-lo a uma variável:

```
return self;
```

## super

Muitas vezes, quando você está sobrescrevendo um método, quer manter o que o método da superclasse está fazendo, e quer que a subclasse faça algo adicional. Para facilitar, há uma diretiva de compilação em Objective-C chamada `super`:

```

- (void)someMethod
{
    [super someMethod];
    [self doMoreStuff];
}

```

Como funciona o `super`? Normalmente, quando se envia uma mensagem a um objeto, a busca pelo método com aquele nome começa na classe do objeto. Se não existir esse método, a busca continua na superclasse do objeto. A busca continuará em toda a hierarquia de herança até que um método apropriado seja encontrado. (Se a busca chegar ao topo da hierarquia e não encontrar nenhum método, uma exceção ocorre.)

Quando se envia uma mensagem ao `super`, uma mensagem está sendo enviada ao `self`, mas a busca pelo método pula a classe do objeto e começa pela superclasse. No caso do inicializador designado de `BNRItem`, você envia a mensagem `init` ao `super`. Isso chama a implementação de `init` da `NSObject`.

## Confirmação do sucesso da inicialização

Agora, vejamos a próxima linha, onde você confirma o que o inicializador da superclasse retornou. Se uma mensagem do inicializador falhar, ele retornará `nil`. Portanto, seria uma boa ideia salvar o valor de retorno do inicializador da superclasse na variável `self` e confirmar que não seja `nil` antes de fazer qualquer outra inicialização.

## Variáveis de instância em inicializadores

Agora chegamos ao centro desse método, onde as variáveis de instância recebem valores. Anteriormente, falamos para você não acessar variáveis de instância diretamente e usar métodos acessores. Agora, pedimos que você quebre essa regra ao escrever inicializadores.

Enquanto um inicializador está sendo executado, o objeto está sendo criado, e você não tem como ter certeza de que todas as suas variáveis de instância foram definidas para valores utilizáveis. Ao escrever um método, você geralmente pressupõe que todas as variáveis de instância de um objeto foram definidas para valores utilizáveis. Assim, chamar um método (como um acessor) no momento, quando esse pode não ser o caso, não é seguro. Na Big Nerd Ranch, nós normalmente definimos as variáveis de instância diretamente nos inicializadores, em vez de chamarmos métodos acessores.

Alguns programadores de Objective-C muito bons utilizam acessores em inicializadores. Eles defendem que, se o acessor fizer algo complicado, você vai querer esse código em exatamente um lugar; substituí-lo em seu inicializador não é bom. Nós não somos totalmente dedicados a nenhuma abordagem, mas neste livro, definiremos as variáveis de instância diretamente nos inicializadores.

## Outros inicializadores e a cadeia de inicializadores

Vamos implementar o segundo inicializador para a classe `BNRItem`. Nessa definição do inicializador, você não vai replicar o código no inicializador designado. Em vez disso, esse inicializador simplesmente chamará o inicializador designado, passando a informação que recebeu para `_itemName` e os valores padrão para os outros argumentos.

No `BNRItem.m`, implemente `initWithItemName:`:

```
- (instancetype)initWithItemName:(NSString *)name
{
    return [self initWithItemName:name
                      valueInDollars:0
                     serialNumber:@""];
}
```

A classe `BNRItem` já tem um terceiro inicializador – `init`, o qual ela herdou da `NSObject`. Se `init` for usado para inicializar uma instância de `BNRItem`, nada do que você colocar no inicializador designado acontecerá. Portanto, você deve sobrescrever `init` na `BNRItem` para vincular ao inicializador designado da `BNRItem`.

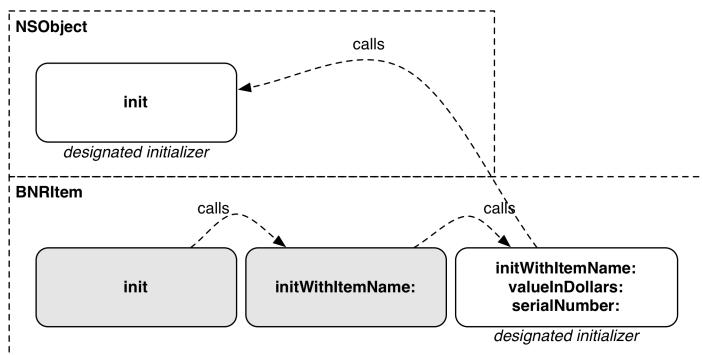
No `BNRItem.m`, sobrescreva `init` para chamar `initWithItemName:`, passando um valor padrão para o nome do item.

```
- (instancetype)init
{
    return [self initWithItemName:@"Item"];
}
```

Agora, quando `init` for enviado para uma instância de `BNRItem`, o método chamará `initWithItemName:` com um valor padrão para `_itemName`, que chamará o inicializador designado `initWithItemName:valueInDollars:serialNumber:` com valores padrão para `_valueInDollars` e `_serialNumber`.

Os relacionamentos entre os inicializadores da `BNRItem` são mostrados na Figure 2.17; os inicializadores designados estão em branco, e os inicializadores adicionais, em cinza.

Figure 2.17 Uma cadeia de inicializadores



Utilizar inicializadores em cadeia reduz a possibilidade de erro e facilita a manutenção do código. O programador que criou a classe deixa claro qual inicializador é o designado. Você escreve apenas a parte essencial do inicializador quando estiver no inicializador designado, e outros métodos de inicialização simplesmente chamarão o inicializador designado (direta ou indiretamente) com valores padrão.

Vamos criar algumas regras simples para os inicializadores a partir destas ideias.

- Uma classe herda todos os inicializadores de sua superclasse, e pode adicionar quantos mais quiser para seus próprios fins.
- Cada classe escolhe um inicializador como seu *inicializador designado*.
- O inicializador designado chama o inicializador designado da superclasse (direta ou indiretamente) antes de qualquer outra coisa.
- Quaisquer outros inicializadores chamam o inicializador designado da classe (direta ou indiretamente).
- Se uma classe declara um inicializador designado diferente daquele da sua superclasse, o inicializador designado da superclasse deve ser sobreescrito para chamar o novo inicializador designado (direta ou indiretamente).

## Utilização de inicializadores

Agora que você tem um inicializador designado para **BNRItem**, você pode utilizá-lo, em vez de definir variáveis de instância individualmente.

No `main.m`, remova a criação da única instância de **BNRItem** e as três mensagens setter. Em seguida, adicione o código que cria uma instância e defina suas variáveis de instância usando o inicializador designado.

```
#import <Foundation/Foundation.h>
#import "BNRItem.h"

int main (int argc, const char * argv[])
{
    @autoreleasepool {

        NSMutableArray *items = [[NSMutableArray alloc] init];

        [items addObject:@"One"];
        [items addObject:@"Two"];
        [items addObject:@"Three"];
        [items insertObject:@"Zero" atIndex:0];

        // For every item in the items array ...
        for (NSString *item in items) {
            // ... print a description of the current item
            NSLog(@"%@", item);
        }

        BNRItem *item = [[BNRItem alloc] init];

        item.itemName = @"Red Sofa";
        item.serialNumber = @"A1B2C";
        item.valueInDollars = 100;

        BNRItem *item = [[BNRItem alloc] initWithitemName:@"Red Sofa"
                                                valueInDollars:100
                                                serialNumber:@"A1B2C"];

        NSLog(@"%@", item);

        items = nil;
    }
    return 0;
}
```

Compile e execute o aplicativo. Observe que o console agora exibe uma única instância de **BNRItem** que foi instanciada com os valores passados ao inicializador designado da classe **BNRItem**.

Vamos verificar se os outros dois inicializadores funcionam conforme o esperado. No `main.m`, crie duas outras instâncias de **BNRItem** usando `initWithitemName: e init`.

```
...
BNRItem *item = [[BNRItem alloc] initWithitemName:@"Red Sofa"
                                         valueInDollars:100
                                         serialNumber:@"A1B2C"];
NSLog(@"%@", item);

BNRItem *itemWithname = [[BNRItem alloc] initWithitemName:@"Blue Sofa"];
NSLog(@"%@", itemWithname);

BNRItem *itemWithNoName = [[BNRItem alloc] init];
NSLog(@"%@", itemWithNoName);

items = nil;
}
return 0;
}
```

Compile e execute o aplicativo, e verifique o console para confirmar que a cadeia de inicialização de **BNRItem** esteja funcionando.

Figure 2.18 Três inicializadores em funcionamento

```
2013-12-08 18:55:18.620 RandomItems[4302:303] Zero
2013-12-08 18:55:18.622 RandomItems[4302:303] One
2013-12-08 18:55:18.623 RandomItems[4302:303] Two
2013-12-08 18:55:18.623 RandomItems[4302:303] Three
2013-12-08 18:55:18.628 RandomItems[4302:303] Red Sofa (A1B2C): Worth $100, recorded on 2013-12-08 23:55:18 +0000
2013-12-08 18:55:18.629 RandomItems[4302:303] Blue Sofa (): Worth $0, recorded on 2013-12-08 23:55:18 +0000
2013-12-08 18:55:18.629 RandomItems[4302:303] Item (): Worth $0, recorded on 2013-12-08 23:55:18 +0000
Program ended with exit code: 0
```

Há apenas mais uma coisa a ser feita para concluir a classe **BNRItem**. Você vai escrever um método que cria uma instância e a inicializa com valores aleatórios. Esse método será um método de classe.

## Métodos de classe

Os métodos de classe normalmente criam novas instâncias da classe ou recuperam alguma propriedade global da classe. Os métodos de classe não operam em uma instância nem têm acesso às variáveis de instância.

Sintaticamente, a diferença entre os métodos de classe e os métodos de instância está no primeiro caractere da respectiva declaração. O método de instância usa o caractere - logo antes do tipo de retorno, e o método de classe usa o caractere +.

No **BNRItem.h**, declare um método de classe que crie um item aleatório.

```
@interface BNRItem : NSObject
{
    NSString *_itemName;
    NSString *_serialNumber;
    int _valueInDollars;
    NSDate *_dateCreated;
}

+ (instancetype)randomItem;

- (instancetype)initWithitemName:(NSString *)name
                           valueInDollars:(int)value
                            serialNumber:(NSString *)sNumber;
```

Observe a ordem das declarações no arquivo de cabeçalho. As variáveis de instância vêm primeiro, seguidas pelos métodos de classe, seguidos pelos inicializadores, seguidos por qualquer outro método de instância. Essa convenção facilita a leitura dos arquivos de cabeçalho.

No **BNRItem.m**, implemente **randomItem** para criar, configurar e retornar uma instância de **BNRItem**. (Certifique-se de que o método esteja entre `@implementation` e `@end`.)

```
+ (instancetype)randomItem
{
    // Create an immutable array of three adjectives
    NSArray *randomAdjectiveList = @[@"Fluffy", @"Rusty", @"Shiny"];

    // Create an immutable array of three nouns
    NSArray *randomNounList = @[@"Bear", @"Spork", @"Mac"];

    // Get the index of a random adjective/noun from the lists
    // Note: The % operator, called the modulo operator, gives
    // you the remainder. So adjectiveIndex is a random number
    // from 0 to 2 inclusive.
    NSInteger adjectiveIndex = arc4random() % [randomAdjectiveList count];
    NSInteger nounIndex = arc4random() % [randomNounList count];

    // Note that NSInteger is not an object, but a type definition
    // for "long"

    NSString *randomName = [NSString stringWithFormat:@"%@ %@",

        [randomAdjectiveList objectAtIndex:adjectiveIndex],
        [randomNounList objectAtIndex:nounIndex]];

    int randomValue = arc4random() % 100;

    NSString *randomSerialNumber = [NSString stringWithFormat:@"%@%c%c%c%c",
        '0' + arc4random() % 10,
        'A' + arc4random() % 26,
        '0' + arc4random() % 10,
        'A' + arc4random() % 26,
        '0' + arc4random() % 10];

    BNRItem *newItem = [[self alloc] initWithitemName:randomName
                                              valueInDollars:randomValue
                                               serialNumber:randomSerialNumber];

    return newItem;
}
```

Primeiro, no início deste método, observe a sintaxe para a criação de dois arrays `randomAdjectiveList` e `randomNounList` – o símbolo @ seguido por colchetes. Dentro dos colchetes está uma lista de objetos limitados por vírgulas que preencherão o array. (Neste caso, os objetos são instâncias de `NSString`.) Essa sintaxe é a forma abreviada para a criação de instâncias de `NSArray`. Observe que ela sempre cria um array imutável. Você só pode usar essa forma abreviada se não precisar que o array resultante seja mutável.

Depois de criar os arrays, `randomItem` cria uma string com base em um adjetivo e um substantivo aleatórios, um valor inteiro aleatório e outra string com base em números e letras aleatórios.

Por fim, o método cria uma instância de `BNRItem` e envia a mensagem do inicializador designado a ela com esses objetos criados aleatoriamente e com o `int` como parâmetros.

Nesse método, você também usou `stringWithFormat:`, o qual é um método de classe de `NSString`. Essa mensagem é enviada diretamente à classe `NSString`, e o método retorna uma instância de `NSString` com os parâmetros passados. No Objective-C, os métodos de classe que retornam um objeto do seu tipo (como `stringWithFormat:` e `randomItem`) são conhecidos como *métodos de conveniência*.

Observe o uso de `self` em `randomItem`. Como `randomItem` é um método de classe, `self` refere-se à própria classe `BNRItem` e não a uma instância. Os métodos de classe devem usar `self` nos métodos de conveniência no lugar do nome da classe, para que a subclasse possa receber a mesma mensagem. Nesse caso, se você criar uma subclasse de `BNRItem` chamada `BNRToxicWasteItem`, poderá fazer o seguinte:

```
BNRToxicWasteItem *item = [BNRToxicWasteItem randomItem];
```

## Teste de subclasse

Para a última versão do `RandomItems` neste capítulo, você vai preencher o array de `items` com 10 instâncias de `BNRItem` criadas aleatoriamente. Depois, você executará um loop no array e registrará cada item (Figure 2.19).

Figure 2.19 Itens aleatórios

```
2013-10-29 18:17:42.880 RandomItems[64653:303] Rusty Spork (8Q2U8): Worth $73, recorded on 2013-10-29 22:17:42 +0000
2013-10-29 18:17:42.880 RandomItems[64653:303] Shiny Spork (5Y2V3): Worth $40, recorded on 2013-10-29 22:17:42 +0000
2013-10-29 18:17:42.881 RandomItems[64653:303] Rusty Spork (2F9Z7): Worth $40, recorded on 2013-10-29 22:17:42 +0000
2013-10-29 18:17:42.881 RandomItems[64653:303] Rusty Bear (8G5V6): Worth $99, recorded on 2013-10-29 22:17:42 +0000
2013-10-29 18:17:42.881 RandomItems[64653:303] Shiny Spork (3P9B1): Worth $10, recorded on 2013-10-29 22:17:42 +0000
2013-10-29 18:17:42.882 RandomItems[64653:303] Rusty Mac (6R5C1): Worth $93, recorded on 2013-10-29 22:17:42 +0000
2013-10-29 18:17:42.882 RandomItems[64653:303] Fluffy Spork (3E400): Worth $1, recorded on 2013-10-29 22:17:42 +0000
2013-10-29 18:17:42.882 RandomItems[64653:303] Fluffy Mac (3A6T4): Worth $30, recorded on 2013-10-29 22:17:42 +0000
2013-10-29 18:17:42.883 RandomItems[64653:303] Shiny Spork (8S3I1): Worth $77, recorded on 2013-10-29 22:17:42 +0000
2013-10-29 18:17:42.883 RandomItems[64653:303] Rusty Spork (4F6F9): Worth $65, recorded on 2013-10-29 22:17:42 +0000
Program ended with exit code: 0
```

No `main.m`, exclua todo o código, exceto de criação e destruição do array de `items`. Agora, adicione 10 instâncias aleatórias de `BNRItem` ao array e registre-as.

```

int main (int argc, const char * argv[])
{
    @autoreleasepool {
        NSMutableArray *items = [[NSMutableArray alloc] init];

        [items addObject:@"One"];
        [items addObject:@"Two"];
        [items addObject:@"Three"];
        [items insertObject:@"Zero" atIndex:0];

        // For every item in the items array ...
        for (NSString *item in items) {
            // ... print a description of the current item
            NSLog(@"%@", item);
        }

        BNRItem *item = [[BNRItem alloc] initWithItemName:@"Red Sofa"
                                                 valueInDollars:100
                                                 serialNumber:@"A1B2C"];

        NSLog(@"%@", item);

        BNRItem *itemWithName = [[BNRItem alloc] initWithItemName:@"Blue Sofa"];
        NSLog(@"%@", itemWithName);

        BNRItem *itemWithNoName = [[BNRItem alloc] init];
        NSLog(@"%@", itemWithNoName);

        for (int i = 0; i < 10; i++) {
            BNRItem *item = [BNRItem randomItem];
            [items addObject:item];
        }

        for (BNRItem *item in items) {
            NSLog(@"%@", item);
        }

        items = nil;
    }
    return 0;
}

```

Observe que você não utiliza a enumeração rápida no primeiro loop, pois está adicionando ao array no loop.

Compile e execute o aplicativo, e verifique os resultados no console.

## Mais informações sobre NSArray e NSMutableArray

Você utilizará arrays frequentemente durante o desenvolvimento de aplicativos iOS, portanto, vamos dar uma olhada em mais alguns detalhes relacionados a arrays.

Um array do Objective-C pode conter objetos de diferentes tipos. Por exemplo, embora o array de `items` contenha apenas instâncias de `BNRItem` no momento, você poderia adicionar uma instância de `NSDate` ou de qualquer outro objeto do Objective-C. Isso é algo diferente da maioria das linguagens fortemente tipadas, nas quais um array pode armazenar apenas objetos de um tipo único.

Os arrays do Objective-C podem reter apenas referências a objetos do Objective-C. Não é possível adicionar dados primitivos e estruturas de C a um array do Objective-C. Caso necessite adicionar dados primitivos ou estruturas de C, você pode “empacotá-los” em objetos do Objective-C escritos para essa finalidade, incluindo `NSNumber`, `NSValue` e `NSData`.

Observe que não é possível adicionar `nil` a um array. Se precisar adicionar “lugares” a um array, você deve usar `NSNull`. `NSNull` é uma classe cuja instância serve para substituir `nil` e é utilizada para essa tarefa, especificamente.

```
[items addObject:[NSNull null]];
```

Ao acessar membros de um array, você utilizou a mensagem **objectAtIndex:** com o índice do objeto que deseja retornar. Isso, como muitos outros elementos do Objective-C, é bastante prolixo. Assim, existe uma sintaxe abreviada para acessar os membros de um array:

```
NSString *foo = items[0];
```

Esta linha de código é equivalente a enviar **objectAtIndex:** para **items**.

```
NSString *foo = [items objectAtIndex:0];
```

No **BNRItem.m**, atualize **randomItem** para utilizar essa sintaxe quando criar o nome aleatório.

```
+ (instancetype)randomItem
{
    ...
    NSString *randomName = [NSString stringWithFormat:@"%@ %@",  

                            {randomAdjectiveList objectAtIndex:adjectiveIndex},  

                            {randomNounList objectAtIndex:nounIndex}];

    NSString *randomName = [NSString stringWithFormat:@"%@ %@",  

                            randomAdjectiveList[adjectiveIndex],  

                            randomNounList[nounIndex]];

    int randomValue = arc4random() % 100;

    ...
    return newItem;
}
```

Compile e execute para confirmar que o programa funciona da mesma forma que antes.

Os colchetes aninhados que resultam podem confundir, pois eles são usados de duas maneiras distintas: um dos usos envia uma mensagem e o outro acessa os itens de um array. Algumas vezes, pode ser mais claro continuar com o envio da mensagem digitada. Outras vezes, é bom evitar digitar o exaustivo **objectAtIndex:**.

Independentemente da sintaxe que você usar, é importante entender que não há diferença em seu aplicativo: o compilador transforma a sintaxe abreviada em um código que envia a mensagem **objectAtIndex:**.

Em uma **NSMutableArray**, você pode usar uma sintaxe abreviada semelhante para adicionar e substituir objetos.

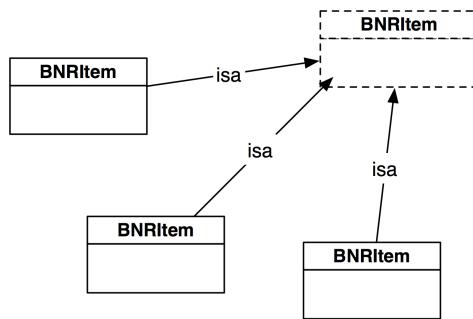
```
NSMutableArray *items = [[NSMutableArray alloc] init];
items[0] = @"A"; // Add @"A"
items[1] = @"B"; // Add @"B"
items[0] = @"C"; // Replace @"A" with @"C"
```

Estas linhas equivalem a enviar mensagens **insertObjectAtIndex:** e **replaceObjectAtIndex withObject:** para **items**.

## Exceções e seletores não reconhecidos

No tempo de execução, quando uma mensagem é enviada a um objeto, esse objeto vai até a classe que a criou e diz: “Recebi essa mensagem. Execute o código para o método correspondente.” Isso é diferente da maioria das linguagens compiladas, em que o método que será executado é determinado durante a compilação.

Como um objeto sabe qual foi a classe que o criou? Ele usa o ponteiro **isa**. Todos os objetos têm uma variável de instância chamada **isa**. Quando um objeto é criado, a classe define a variável de instância **isa** do objeto retornado para apontar de volta para essa classe (Figure 2.20). Chamamos isso de ponteiro **isa** porque o objeto “é uma” instância dessa classe. Embora você provavelmente nunca usará o ponteiro **isa** explicitamente, a sua existência é uma das grandes fontes de poder do Objective-C.

Figure 2.20 O ponteiro `isa`

Um objeto somente responde a uma mensagem se sua classe (apontada pelo ponteiro `isa`) implementa o método associado. Como isso ocorre durante o tempo de execução, o Xcode nem sempre consegue entender durante a compilação (durante o build do aplicativo) se um objeto responderá a uma mensagem. O Xcode exibirá um erro caso ache que você está enviando uma mensagem a um objeto que não pode responder, mas se ele não tiver certeza, permitirá a compilação (build) do aplicativo.

Se, por algum motivo (e podem haver muitos), você acabar enviando uma mensagem a um objeto que não responde, seu aplicativo sofrerá uma *exceção*. As exceções também são conhecidas como *erros em tempo de execução* porque ocorrem durante a execução do aplicativo, diferentemente dos *erros em tempo de compilação* que aparecem durante o build do aplicativo, ou compilação.

Para praticar o que fazer com exceções, você vai causar uma em RandomItems.

No `main.m`, obtenha o último item do array usando o método `lastObject` de `NSArray`. Depois, envie ao item uma mensagem que ele não vai entender:

```

#import <Foundation/Foundation.h>
#import "BNRItem.h"

int main (int argc, const char * argv[])
{
    @autoreleasepool {
        NSMutableArray *items = [[NSMutableArray alloc] init];

        for (int i = 0; i < 10; i++) {
            BNRItem *item = [BNRItem randomItem];
            [items addObject:item];
        }

        id lastObj = [items lastObject];

        // lastObj is an instance of BNRItem and will not understand the count message
        [lastObj count];

        for (BNRItem *item in items) {
            NSLog(@"%@", item);
        }

        items = nil;
    }
    return 0;
}

```

Compile e execute o aplicativo. Seu aplicativo vai compilar, iniciar a execução e depois parar. Verifique o seu console e procure a linha semelhante à seguinte:

```

2014-01-19 12:23:47.990 RandomItems[10288:707] ***
Terminating app due to uncaught exception 'NSInvalidArgumentException', reason:
'-[BNRItem count]: unrecognized selector sent to instance 0x100117280'

```

Essa é a aparência de uma exceção. O que ela está dizendo exatamente? Primeiro ela informa a data, a hora e o nome do aplicativo. Você pode ignorar essa informação e se focar no que vem depois de “\*\*\*”. Essa linha diz que ocorreu uma exceção e o motivo para isso.

O motivo é a informação mais importante que a exceção nos dá. Aqui, o motivo informa que um *seletor não reconhecido* foi enviado a uma instância. Você sabe que seletor é o mesmo que mensagem. Você enviou uma mensagem a um objeto, e o objeto não implementou o método.

O tipo de destinatário e o nome da mensagem também são informados nessa saída, o que facilita a depuração. Uma instância de `BNRItem` recebeu a mensagem `count`. O - no início nos mostra que o destinatário era uma instância de `BNRItem`. Um + indicaria que o destinatário era a própria classe.

Há duas lições importantes aqui. Primeiro, sempre verifique o console quando seu aplicativo parar ou travar; os erros em tempo de execução (exceções) são tão importantes quanto aqueles que ocorrem na compilação. Segundo, lembre-se que um *seletor não reconhecido* significa que a mensagem que você está enviando não é implementada pelo destinatário. Você cometerá esse erro mais de uma vez, então irá identificá-lo rapidamente.

Algumas linguagens usam blocos try e catch para tratar exceções. Embora o Objective-C tenha essa capacidade, não a usamos com muita frequência no código do aplicativo. Geralmente, a exceção é um erro do programador e deve ser corrigido no código em vez de ser tratado no tempo de execução.

No `main.m`, remova o código que está causando a exceção.

```
for (int i = 0; i < 10; i++) {
    BNRItem *item = [BNRItem randomItem];
    [items addObject:p];
}

id lastObj = [items lastObject];
[lastObj count];

for (BNRItem *item in items) {
    NSLog(@"%@", item);
}
```

## Desafios

A maioria dos capítulos deste livro termina com pelo menos um desafio para incentivar você a ir mais longe e provar a si mesmo que aprendeu aquilo que foi ensinado. Sugerimos que você tente resolver o máximo de desafios que puder, para consolidar os seus conhecimentos, e não apenas *aprender* programação de iOS conosco, mas também *implementar* programação de iOS por conta própria.

Os desafios têm três níveis de dificuldade:

- Os desafios de bronze normalmente pedem que você faça algo bastante parecido com o que foi feito no capítulo. Esses desafios reforçam o que você aprendeu no capítulo e fazem você inserir um código parecido sem que esteja olhando para ele. A prática leva à perfeição.
- Os desafios de prata exigem que você explore mais e reflita mais sobre os conceitos. Você precisará usar métodos, classes e propriedades que não viu antes, mas as tarefas ainda são semelhantes ao que você fez no capítulo.
- Os desafios de ouro são difíceis e podem demandar várias horas. Eles exigem que você tenha compreendido os conceitos do capítulo, e reflita e tente achar sozinho a solução para determinados problemas. Esses desafios irão preparar você para o trabalho de programação para iOS na vida real.

Antes de iniciar qualquer desafio, *sempre faça uma cópia do seu diretório de projeto no Finder e faça o exercício usando essa cópia*. Muitos capítulos baseiam-se em capítulos anteriores, e usar a cópia para fazer o exercício permite que você vá progredindo conforme avança no livro.

### Desafio de bronze: detecção de bug

Crie um bug no seu programa, solicitando o décimo-primeiro item do array. Execute-o e observe a exceção que ocorre.

## Desafio de prata: outro inicializador

Crie outro método inicializador para a classe **BNRItem**. Esse inicializador *não* é o inicializador designado de **BNRItem**. Ele leva uma instância de **NSString** que identifica a `itemName` do item e uma instância de **NSString** que identifica a `serialNumber`.

## Desafio de ouro: outra classe

Crie uma subclasse de **BNRItem** chamada **BNRContainer**. Uma instância de **BNRContainer** deve ter um array de **subitems** que contenha instâncias de **BNRItem**. A exibição da descrição do objeto de uma **BNRContainer** deve mostrar a você o nome do recipiente, seu valor em dólares (soma de todos os itens no recipiente mais o valor do próprio recipiente) e uma lista de cada instância de **BNRItem** que ele contém. Uma classe **BNRContainer** escrita corretamente pode conter instâncias de **BNRContainer**. Ela também pode reportar corretamente o seu valor total e todos os itens que contém.

## Você está mais curioso?

Além dos Desafios, muitos capítulos terminam com uma ou mais seções “Para os mais curiosos”. Essas seções oferecem explicações mais profundas ou informações adicionais sobre os tópicos do capítulo. Os conhecimentos disponíveis nessas seções não são indispensáveis para o seu objetivo, mas esperamos que sejam interessantes e úteis para você.

## Para os mais curiosos: nomes de classe

Em aplicativos simples como o `RandomItems`, usamos apenas um pequeno número de classes. Conforme os aplicativos ficam maiores e mais complexos, o número de classes aumenta. Em algum momento, você terá duas classes que poderiam facilmente ter o mesmo nome. Isso não é bom. Se duas classes tiverem o mesmo nome, seu programa não conseguirá saber qual delas ele deve usar. Essa situação é conhecida como *colisão de namespace*.

Outras linguagens resolvem esse problema declarando as classes dentro de um *namespace*. Pense no namespace como um grupo ao qual as classes pertencem. Para usar uma classe nessas linguagens, é preciso especificar o nome da classe e o namespace.

O Objective-C não tem a noção de namespaces. Em vez disso, os nomes de classes são prefixados com duas ou três letras para diferenciá-los. Por exemplo, neste exercício, a classe recebeu o nome **BNRItem** em vez de **Item**.

Os programadores de Objective-C mais descolados sempre prefixam suas classes. O prefixo está geralmente relacionado ao nome do aplicativo que você está desenvolvendo ou à biblioteca ao qual pertence. Por exemplo, se eu estivesse escrevendo um aplicativo chamado “MovieViewer”, prefixaria todas as classes com **MOV**. As classes que você usará em vários projetos normalmente carregam um prefixo relacionado ao seu nome (**CBK**), o nome da sua empresa (**BNR**) ou uma biblioteca portátil (uma biblioteca relacionada a mapas poderia usar **MAP**).

Note que as classes da Apple também têm prefixos. As classes da Apple são organizadas em frameworks, e cada framework tem seu próprio prefixo. Por exemplo, a classe **UILabel** pertence ao framework **UIKit**. As classes **NSArray** e **NSString** pertencem ao framework **Foundation**. (**NS** quer dizer NeXTSTEP, a plataforma para a qual essas classes foram criadas originalmente.)

Para as suas classes, você deve utilizar prefixos de três letras. Os prefixos de duas letras são reservados pela Apple para serem usados em classes de framework. Embora nada o impeça de criar uma classe com um prefixo de duas letras, você deveria utilizar prefixos de três letras para eliminar a possibilidade de colisão de namespace com as classes atuais e futuras da Apple.

## Para os mais curiosos: #import e @import

#

Quando o Objective-C era recente, o sistema não era enviado com muitas classes. No entanto, com o tempo, havia tantas classes que foi preciso organizá-las em frameworks. Em seu código-fonte, você normalmente iria #import(ar) o arquivo de cabeçalho mestre para um framework:

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

E esse arquivo iria #import todos os cabeçalhos naquele framework, assim:

```
#import <Foundation/NSArray.h>
#import <Foundation/NSAutoreleasePool.h>
#import <Foundation/NSBundle.h>
#import <Foundation/NSByteOrder.h>
#import <Foundation/NSCalendar.h>
#import <Foundation/NSCharacterSet.h>
...
```

Depois, você vincularia explicitamente esse framework em seu programa durante a compilação.

Isso foi fácil de implementar; estava utilizando o pré-processador em C existente para copiar todos os cabeçalhos no arquivo que estava prestes a ser compilado.

Essa abordagem funcionou muito bem durante cerca de uma década. Então, à medida que mais classes foram adicionadas aos frameworks e mais frameworks eram usados em cada projeto, percebemos que o compilador estava gastando a maior parte de seu tempo analisando e processando esses mesmos cabeçalhos padrão, repetidamente. Assim, o *arquivo de cabeçalho pré-compilado* foi adicionado a todo projeto. A primeira vez que você compilasse seu projeto, os cabeçalhos listados nesse arquivo seriam compilados uma vez e o resultado seria guardado em cache. Ter esse grupo de cabeçalhos pré-compilados fez com que a compilação de todos os outros arquivos ficasse muito mais rápida. O projeto que você acabou de criar tem o arquivo RandomItems-prefix.pch, e ele força o sistema de compilação a pré-compilar os cabeçalhos para o framework Foundation:

```
#ifdef __OBJC__
    #import <Foundation/Foundation.h>
#endif
```

Você ainda precisava vincular explicitamente esse framework em seu programa durante a compilação.

Isso funcionou muito bem por mais uma década, mas a Apple percebeu recentemente que os desenvolvedores não estavam mantendo seus arquivos .pch de forma eficaz. Assim, eles tornaram o compilador mais inteligente e a diretiva @import foi introduzida:

```
@import Foundation;
```

Isso diz ao compilador: “Estou usando o módulo Foundation. Descubra como fazer isso funcionar.” O compilador tem muita liberdade para otimizar o pré-processamento e o cache de arquivos de cabeçalho. (Isso também elimina a necessidade de vincular o módulo explicitamente no programa – quando o compilador vê o @import, ele faz uma anotação para vincular no módulo apropriado.)

Até o momento em que este livro foi escrito, apenas a Apple pode criar módulos que podem ser usados com @import. Para usar classes e frameworks criados por você, você ainda precisará usar o #import.

Escrevemos este livro no Xcode 5.0, e o #import ainda aparece nos templates de projetos e arquivos, mas estamos certos de que, no futuro próximo, o @import será ubíquo.



# 3

## Gerenciamento de memória com ARC

Neste capítulo, você aprenderá como a memória é gerenciada em iOS e os conceitos que sustentam a *contagem de referência automática*, ou ARC. Vamos começar com algumas noções básicas de memória de aplicativo.

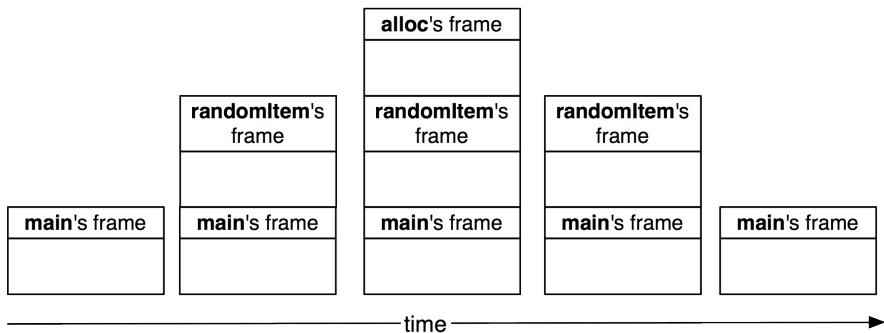
### A pilha

Quando um método (ou função) é executado, um pedaço (chunk) de memória é alocado a partir de uma parte da memória chamada *pilha*. Esse pedaço de memória é chamado de *frame* e armazena os valores para as variáveis declaradas dentro do método. Uma variável declarada dentro de um método chama-se *variável local*.

Quando um aplicativo é iniciado e executa `main()`, o frame para `main()` é colocado na pilha. Quando `main()` chama outro método (ou função), o frame para aquele método é colocado no topo da pilha. Claro, aquele método poderia chamar outro método, e assim por diante, até você ter uma pilha de frames bastante alta. Conforme cada método ou função termina, seu frame é “retirado” do topo da pilha e destruído. Se o método for chamado novamente, um novo frame será alocado e colocado na pilha.

Por exemplo, no aplicativo `RandomItems`, a função `main` executa o método `randomItem` de `BNRItem`, que por sua vez executa `alloc`. A pilha ficaria parecida com a Figure 3.1. Observe que o frame de `main()` continua ativo enquanto os outros métodos são executados, pois ele ainda não terminou de ser executado.

Figure 3.1 Aumento e diminuição da pilha



O método `randomItem` é executado dentro de um loop em `main()`. Com cada iteração do loop, a pilha aumenta e diminui à medida que frames são colocados e retirados da pilha.

### O Heap

Existe outra parte da memória, chamada *heap*, que é separada da pilha. A razão dos nomes “heap” e “pilha” tem a ver com a forma como você os visualiza. A pilha pode ser visualizada como uma pilha ordenada de frames. O heap, por outro lado, é onde todos os objetos do Objective-C residem. Ele é um enorme amontoamento de objetos. Você usa ponteiros para controlar onde esses objetos estão armazenados dentro do heap.

Quando você envia a mensagem `alloc` para uma classe, um pedaço da memória é alocado a partir do heap. Esse pedaço é o seu objeto, e ele inclui espaço para as variáveis de instância do objeto. Uma instância de `BNRItem`

tem cinco variáveis de instância: quatro ponteiros (`isa`, `_itemName`, `_serialNumber` e `_dateCreated`) e um `int` (`_valueInDollars`). Assim, o pedaço de memória que é alocado inclui espaço para um `int` e quatro ponteiros. Esses ponteiros armazenam os endereços de outros objetos no heap.

Um aplicativo iOS cria objetos na inicialização e, normalmente, continuará a criar objetos durante todo o tempo em que o aplicativo estiver em execução. Se a memória heap fosse infinita, o aplicativo poderia criar todos os objetos que quisesse e faria com que existissem para a execução inteira do aplicativo.

Mas um aplicativo tem um limite de memória heap e a memória de um dispositivo iOS é especialmente limitada. Portanto, este recurso deve ser gerenciado: É importante destruir os objetos que não são mais necessários para liberar espaço na memória heap que poderá ser reutilizado para criar novos objetos. Por outro lado, é essencial não destruir os objetos que ainda são necessários.

## ARC e gerenciamento de memória

A boa notícia é que você não precisa controlar quais objetos devem viver ou morrer. O gerenciamento de memória do seu aplicativo é mantido para você pela ARC, que significa contagem de referência automática. Todos os aplicativos deste livro usarão ARC. Antes da ARC estar disponível, os aplicativos usavam a *contagem de referência manual*. Há mais informações sobre a contagem de referência manual no final do capítulo.

Na maioria dos casos, pode-se contar com a ARC para gerenciar a memória do aplicativo automaticamente. Contudo, é importante entender os conceitos por trás disso para saber como intervir, quando necessário. Vamos começar com a ideia de propriedade de objeto.

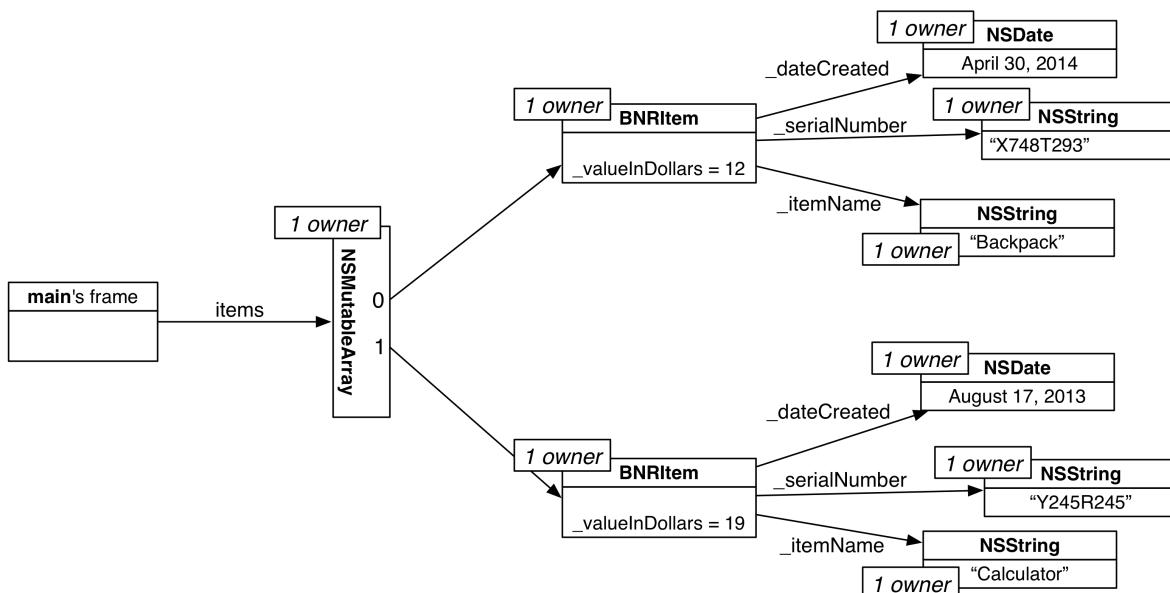
## Variáveis ponteiros e propriedade de objeto

Variáveis ponteiros indicam a *propriedade* dos objetos para os quais apontam.

- Quando um método (ou função) tem uma variável local que aponta para um objeto, diz-se que essa variável *possui* o objeto sendo apontado.
- Quando um objeto tem uma variável de instância que aponta para outro objeto, diz-se que o objeto com o ponteiro *possui* o objeto sendo apontado.

Pense em seu aplicativo `RandomItems`. Nesse aplicativo, uma instância de `NSMutableArray` é criada na função `main()` e, em seguida, 10 instâncias `BNRItem` são adicionadas a ela. A Figure 3.2 mostra alguns dos objetos no `RandomItems` e os ponteiros que fazem referência a eles.

Figure 3.2 Diagrama de objetos de `RandomItems` (com apenas dois itens)



Dentro de `main()`, a variável local `items` aponta para uma instância de `NSMutableArray`, portanto, `main()` possui essa instância de `NSMutableArray`.

O array, por sua vez, possui as instâncias de `BNRItem`. Um objeto de coleção, assim como uma instância de `NSMutableArray`, tem ponteiros para objetos, em vez de realmente contê-los, e tais ponteiros indicam propriedade: um array sempre possui os objetos que estão “dentro” do array.

Finalmente, cada instância de `BNRItem` possui os objetos que são apontados por suas variáveis de instância.

A ideia de propriedade de objeto é útil para determinar se um objeto será destruído para que sua memória possa ser reutilizada.

- *Um objeto sem nenhum proprietário será destruído.* Um objeto sem proprietário não pode receber mensagens e está isolado e sem utilidade para o aplicativo. Mantê-lo gasta memória valiosa. Isso é chamado de *vazamento de memória*.
- *Um objeto com um ou mais proprietários não será destruído.* Se um objeto é destruído, mas outro objeto ou método ainda tem um ponteiro apontando para ele (ou, mais precisamente, um ponteiro que armazena o endereço onde ele *costumava* ficar), então, você está em uma situação perigosa: enviar uma mensagem por meio desse ponteiro pode fazer com que seu aplicativo falhe. Destruir um objeto que ainda é necessário chama-se *desalocação prematura*. Isso também é conhecido como *ponteiro pendente* ou *referência pendente*.

## Como objetos perdem proprietários

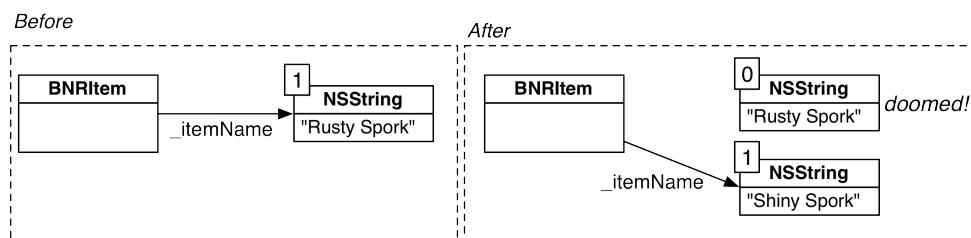
Estas são as maneiras de um objeto perder um proprietário:

- Uma variável que aponta para o objeto é alterada para apontar para outro.
- Uma variável que aponta para o objeto é colocada em `nil`.
- O próprio proprietário do objeto é destruído.
- Um objeto em uma coleção, assim como um array, é removido dessa coleção.

Vamos dar uma olhada em cada uma dessas situações.

### Alteração de um ponteiro

Imagine uma instância de `BNRItem`. A sua variável de instância `_itemName` aponta para uma instância de `NSString`, @"RustySpork" (garfo enferrujado). Se você polisse a ferrugem do garfo, ele se tornaria um garfo brilhante, e você, então, desejaria mudar a `_itemName` de modo a apontar para uma `NSString` diferente.



Quando o valor de `_itemName` muda do endereço da string “Rusty Spork” (garfo enferrujado) para o endereço da string “Shiny Spork” (garfo brilhante), a string “Rusty Spork” perde um proprietário. Se ela não tiver mais nenhum proprietário, então, será destruída.

### Definição de um ponteiro para nil

Definir um ponteiro para `nil` representa a ausência de um objeto. Por exemplo, digamos que você tenha uma instância de `BNRItem` que representa uma televisão. Então, alguém apaga o número de série da televisão. Você

definiria a variável de instância `_serialNumber` para `nil`. A instância `NSString` para a qual `_serialNumber` costumava apontar perde um proprietário.

## O proprietário foi destruído.

Quando um objeto é destruído, os objetos que ele possui perdem um proprietário. Dessa forma, um objeto sendo desalocado pode causar uma sucessão de desalocações de objetos.

Através de suas variáveis locais, um método ou uma função podem ser proprietários de objetos. Quando o método ou a função deixa de ser executada e o seu frame é retirado da pilha, os objetos que possuem perdem um proprietário.

## Remoção de um objeto de uma coleção

Há mais uma maneira importante de um objeto perder um proprietário. Um objeto em um objeto de coleção é de propriedade do objeto de coleção. Ao remover um objeto de um objeto de coleção mutável, como uma instância de `NSMutableArray`, o objeto removido perde um proprietário.

```
[items removeObject:item]; // Object pointed to by item loses an owner
```

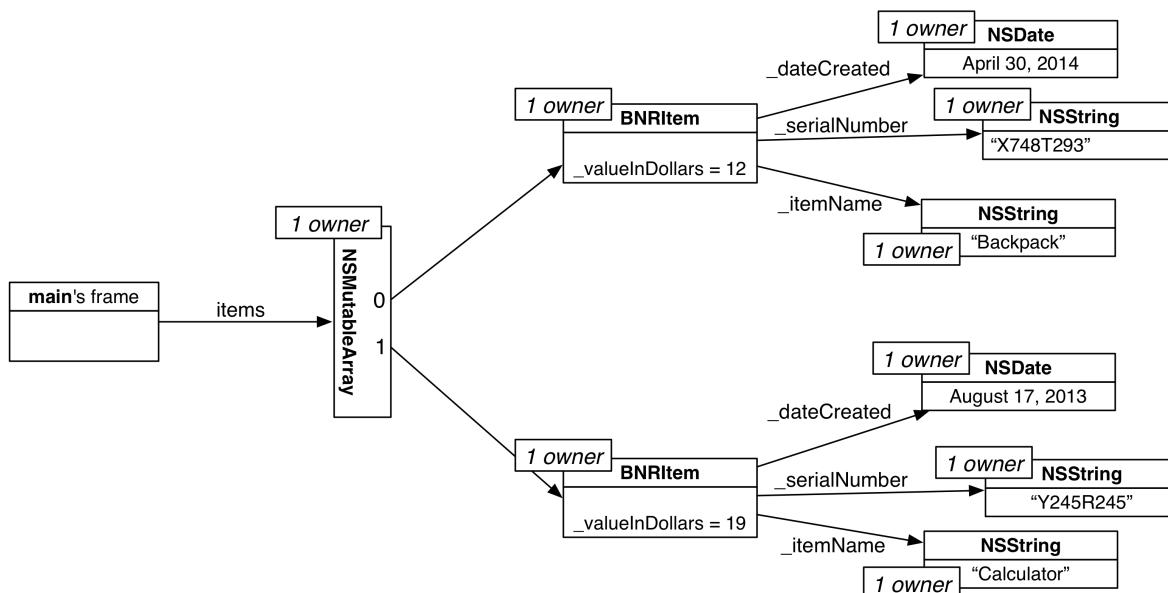
Não esqueça que perder um proprietário por algum desses meios não significa, necessariamente, que o objeto será destruído. Se ainda houver outro ponteiro para o objeto em algum lugar, ele continuará a existir. Quando um objeto perde seu último proprietário, o resultado é morte rápida e certa.

## Cadeias de propriedade

Como objetos são proprietários de objetos, que podem ser proprietários de outros objetos, a destruição de um único objeto pode desencadear uma reação em cadeia de perda de propriedade, destruição de objetos e liberação de espaço de memória.

Há um exemplo disso no `RandomItems`. Veja novamente o diagrama de objetos para esse aplicativo.

Figure 3.3 Objetos e ponteiros no `RandomItems`



No `main.m`, depois de exibir o array, você define a variável `items` para `nil`. Definir `items` para `nil` faz com que o array perca seu único proprietário, portanto, o array é destruído.

Mas a destruição não para por aí. Quando o array é destruído, todos os ponteiros para as instâncias de `BNRItem` são destruídos. Depois que estas variáveis somem, os itens não têm mais nenhum proprietário e, portanto, são todos destruídos.

Finalmente, destruir uma **BNRItem** destrói suas variáveis de instância, o que deixa os objetos apontados por elas sem proprietários. Assim, eles também são destruídos.

Vamos acrescentar um pouco de código, desse modo você pode ver essa destruição em ação. **NSObject** implementa um método **dealloc**, que é enviado para um objeto quando está prestes a ser destruído. Você pode sobrescrever **dealloc** em **BNRItem** para que algo seja exibido no console quando um item for destruído.

No projeto RandomItems, abra **BNRItem.m** e sobrescreva **dealloc**.

```
- (void)dealloc
{
    NSLog(@"Destroyed: %@", self);
}
```

No `main.m`, adicione a linha de código abaixo.

```
NSLog(@"Setting items to nil...");  
items = nil;
```

Compile e execute o aplicativo. Depois da exibição dos itens, você verá a mensagem anunciando que `items` está sendo definido em `nil`. Depois, você verá a destruição de cada `BNRItem` registrada no console.

Ao final, não há mais nenhum objeto ocupando a memória e apenas a função `main` permanece. Toda essa limpeza e reciclagem automática da memória ocorrem como resultado da definição de `items` em `nil`. Este é o poder da ARC.

## Referências fortes e fracas

Dissemos que sempre que uma variável ponteiro aponta para um objeto, esse objeto tem um proprietário e permanecerá ativo. Isso é conhecido como uma *referência forte*.

Uma variável pode, opcionalmente, *não* assumir a propriedade de um objeto para o qual aponta. Uma variável que não assume a propriedade de um objeto é conhecida como uma *referência fraca*.

Uma referência fraca é útil para prevenir um problema chamado *ciclo de referência forte* (também conhecido como *ciclo de retenção*.) Um ciclo de referência forte ocorre quando dois ou mais objetos têm referências fortes um para o outro. Isso não é bom. Quando dois objetos são proprietários um do outro, nunca podem ser destruídos pela ARC. Mesmo se todos os outros objetos no aplicativo liberarem a propriedade destes objetos, eles (e quaisquer objetos que possuam) continuarão a existir dentro de sua bolha de propriedade mútua.

Assim, um ciclo de referência forte é um vazamento de memória, o qual a ARC precisa da sua ajuda para corrigir. Você pode corrigir isso tornando fraca uma das referências.

Vamos inserir um ciclo de referência forte no `RandomItems` para vermos como funciona. Primeiro, você dará a uma instância de `BNRItem` a capacidade de armazenar outra `BNRItem` (para representar algo como uma mochila ou uma bolsa). Além disso, um item saberá que outro item o contém.

No `BNRItem.h`, declare duas variáveis de instância e seus métodos acessores.

```
@interface BNRItem : NSObject  
{  
    NSString *_itemName;  
    NSString *_serialNumber;  
    int _valueInDollars;  
    NSDate *_dateCreated;  
  
    BNRItem *_containedItem;  
    BNRItem *_container;  
}  
  
+ (instancetype)randomItem;  
  
- (instancetype)initWithItemName:(NSString *)name  
                      valueInDollars:(int)value  
                      serialNumber:(NSString *)sNumber;  
  
- (instancetype)initWithItemName:(NSString *)name;  
  
- (void)setContainedItem:(BNRItem *)item;  
- (BNRItem *)containedItem;  
  
- (void)setContainer:(BNRItem *)item;  
- (BNRItem *)container;
```

No `BNRItem.m`, implemente os métodos acessores.

```

- (void)setContainedItem:(BNRItem *)item
{
    _containedItem = item;

    // When given an item to contain, the contained
    // item will be given a pointer to its container
    item.container = self;
}

- (BNRItem *)containedItem
{
    return _containedItem;
}

- (void)setContainer:(BNRItem *)item
{
    _container = item;
}

- (BNRItem *)container
{
    return _container;
}

```

No `main.m`, retire o código que preenche o array com itens aleatórios. Depois, crie dois novos itens, adicione-os ao array e faça com que apontem um para o outro.

```

int main (int argc, const char * argv[])
{
    @autoreleasepool {
        NSMutableArray *items = [[NSMutableArray alloc] init];

        for (int i = 0; i < 10; i++) {
            BNRItem *item = [BNRItem randomItem];
            [items addObject:item];
        }

        BNRItem *backpack = [[BNRItem alloc] initWithItemName:@"Backpack"];
        [items addObject:backpack];

        BNRItem *calculator = [[BNRItem alloc] initWithItemName:@"Calculator"];
        [items addObject:calculator];

        backpack.containedItem = calculator;

        backpack = nil;
        calculator = nil;

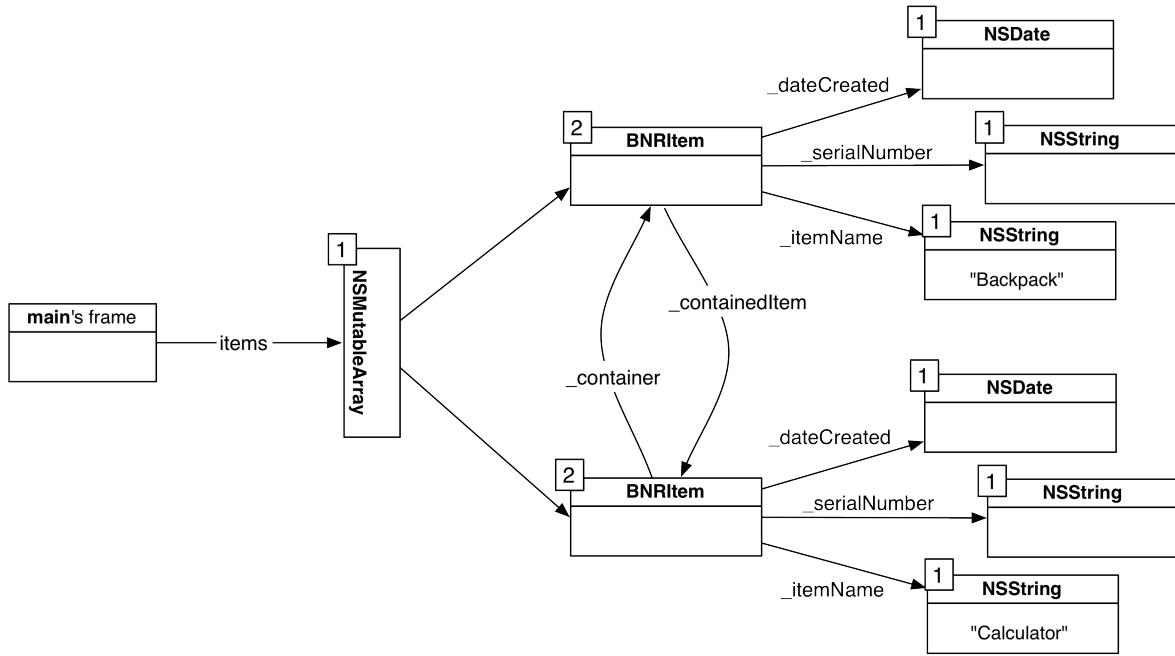
        for (BNRItem *item in items)
            NSLog(@"%@", item);

        NSLog(@"Setting items to nil...");
        items = nil;
    }
    return 0;
}

```

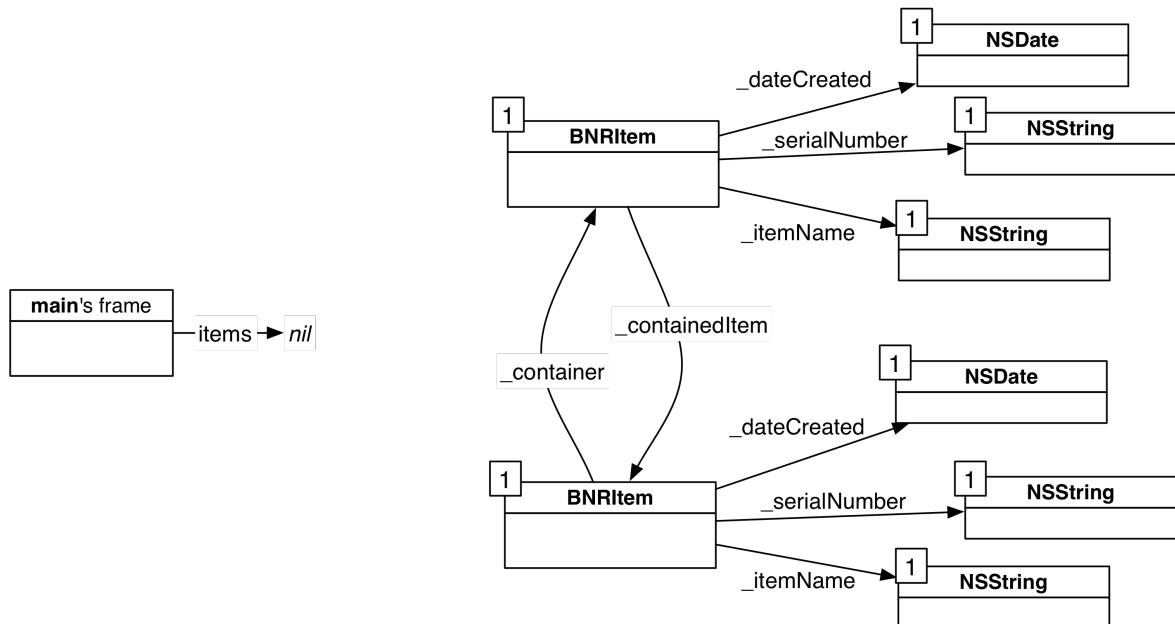
Esta é a aparência do aplicativo agora:

Figure 3.4 RandomItems com ciclo de referência forte



Compile e execute o aplicativo. Dessa vez, você não verá nenhuma mensagem informando a destruição dos objetos **BNRItem**. Este é um ciclo de referência forte: a mochila (backpack) e a calculadora (calculator) têm referências fortes uma à outra, portanto, não há como destruir esses objetos. Figure 3.5 mostra os objetos no aplicativo que ainda estão ocupando memória, depois que `items` foi definida para `nil`.

Figure 3.5 Vazamento de memória!



Os dois itens não podem ser acessados por nenhuma outra parte do aplicativo (neste caso, `main()`), mesmo assim, eles ainda existem, sem fazer nada de útil. Além do mais, como não podem ser destruídos, o mesmo é verdade para os objetos para os quais suas variáveis de instância apontam.

Para corrigir esse problema, um dos ponteiros entre os itens precisa ser uma referência fraca. Para decidir qual deles deve ser fraco, pense nos objetos do ciclo como um relacionamento pai-filho. Nesse relacionamento,

o filho pode pertencer ao pai, mas o pai nunca deve pertencer ao filho. Em nosso ciclo de referência forte, a mochila é o pai e a calculadora é o filho. Assim, a mochila pode manter sua referência forte para a calculadora (a variável de instância `_containedItem`), mas a referência da calculadora para a mochila (a variável de instância `_container`) deve ser fraca.

Para declarar uma variável como uma referência fraca, usamos o atributo `__weak`. No `BNRItem.h`, altere a variável de instância `container` para que seja uma referência fraca.

```
__weak BNRItem *_container;
```

Compile e execute o aplicativo novamente. Desta vez, os objetos são destruídos adequadamente.

A maioria dos ciclos de referências fortes pode ser analisada como um relacionamento pai-filho. Um pai, normalmente, tem uma referência forte para o filho, assim, se um filho precisa de um ponteiro para o pai, esse ponteiro deve ser uma referência fraca, a fim de evitar um ciclo de referência forte.

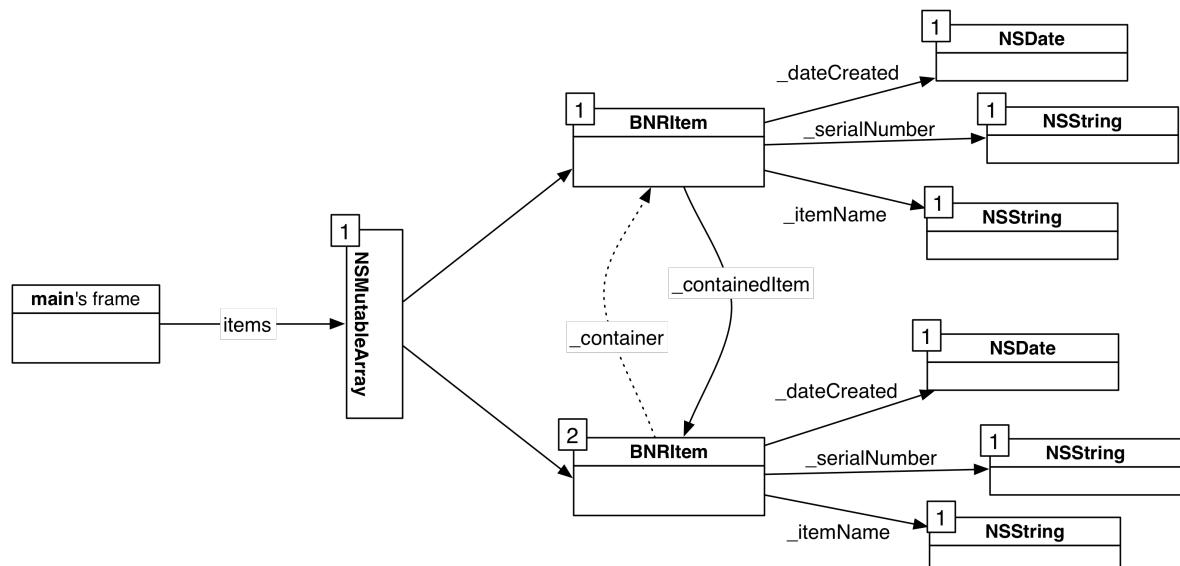
Um filho que tem uma referência forte para o pai do *pai* também gera um ciclo de referência forte. Então, a mesma regra se aplica nessa situação: se um filho precisa de um ponteiro para o pai do pai (ou pai do pai do pai etc.), então, o ponteiro deve ser uma referência fraca.

As ferramentas de desenvolvimento da Apple incluem uma ferramenta `Leaks` para ajudá-lo a encontrar ciclos de referências fortes. Você verá como usar essa ferramenta no Chapter 14.

Uma referência fraca sabe quando o objeto para o qual aponta é destruído e responde definindo-se para `nil`. Dessa forma, se a mochila for destruída, a variável de instância `_container` da calculadora será definida para `nil` automaticamente. Isso é conveniente. Se `_container` não fosse definida para `nil`, destruir o objeto lhe deixaria um ponteiro pendente, o que poderia fazer com que o aplicativo falhasse.

Veja o diagrama atual de `RandomItems`. Observe que a seta que representa a variável ponteiro `container` agora é uma linha pontilhada. A linha pontilhada indica uma referência fraca. Referências fortes são sempre linhas sólidas.

Figure 3.6 RandomItems com ciclo de referência forte evitado



## Propriedades

Toda vez que você declarou uma variável de instância em **BNRItem**, você declarou e implementou um par de métodos acessores. Agora, você vai aprender a usar *propriedades*, uma alternativa conveniente para escrever métodos acessores que economiza muita digitação e torna seus arquivos de classes muito mais fáceis de ler.

### Declaração de propriedades

Uma declaração de propriedade tem a seguinte forma:

```
@property NSString *itemName;
```

Por padrão, ao declarar uma propriedade você obtém três coisas: uma variável de instância e dois acessores para a variável de instância. Veja a Table 3.1, que mostra à esquerda uma classe que não utiliza propriedades e à direita a classe equivalente com propriedades.

Table 3.1 Com e sem propriedades

|            | Sem propriedades                                                                                                                   | Com propriedades                                                         |
|------------|------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------|
| BNRThing.h | <pre>@interface BNRThing : NSObject {     NSString *_name; } - (void)setName:(NSString *)n; - (NSString *)name; @end</pre>         | <pre>@interface BNRThing : NSObject @property NSString *name; @end</pre> |
| BNRThing.m | <pre>@implementation BNRThing - (void)setName:(NSString *)n {     _name = n; } - (NSString *)name {     return _name; } @end</pre> | <pre>@implementation BNRThing @end</pre>                                 |

As duas classes na Table 3.1 são exatamente iguais: cada uma delas tem uma variável de instância para o nome da instância e um setter e um getter para o nome. À esquerda, você mesmo digita estas declarações e variáveis de instância. À direita, você simplesmente declara uma propriedade.

Você vai substituir suas variáveis de instância e os métodos acessores em **BNRItem** por propriedades.

No BNRItem.h, exclua a área da variável de instância e as declarações dos métodos acessores. Em seguida, adicione as declarações de propriedade que os substituem.

```

@interface BNRItem : NSObject
{
    NSString *_itemName;
    NSString *_serialNumber;
    int _valueInDollars;
    NSDate *_dateCreated;

    BNRItem *_containedItem;
    __weak BNRItem *_container;
}

@property BNRItem *containedItem;
@property BNRItem *container;

@property NSString *itemName;
@property NSString *serialNumber;
@property int valueInDollars;
@property NSDate *dateCreated;

+ (instancetype)randomItem;

- (instancetype)initWithItemName:(NSString *)name
                           valueInDollars:(int)value
                            serialNumber:(NSString *)sNumber;

- (instancetype)initWithItemName:(NSString *)name;

-(void)setItemName:(NSString *)str;
-(NSString *)itemName;

-(void)setSerialNumber:(NSString *)str;
-(NSString *)serialNumber;

-(void)setValueInDollars:(int)v;
-(int)valueInDollars;

-(NSDate *)dateCreated;

-(void)setContainedItem:(BNRItem *)item;
-(BNRItem *)containedItem;

-(void)setContainer:(BNRItem *)item;
-(BNRItem *)container;

@end

```

Agora, BNRItem.h ficou muito mais fácil de ler:

```

@interface BNRItem : NSObject

+ (instancetype)randomItem;

- (instancetype)initWithItemName:(NSString *)name
                           valueInDollars:(int)value
                            serialNumber:(NSString *)sNumber;

- (instancetype)initWithItemName:(NSString *)name;

@property BNRItem *containedItem;
@property BNRItem *container;

@property NSString *itemName;
@property NSString *serialNumber;
@property int valueInDollars;
@property NSDate *dateCreated;

@end

```

Observe que os nomes das propriedades são os nomes das variáveis de instância sem o sublinhado. No entanto, a variável de instância gerada por uma propriedade tem um sublinhado no nome.

Vejamos um exemplo. Quando você declarou a propriedade chamada `itemName`, você obteve uma variável de instância chamada `_itemName`, um método getter chamado `itemName` e um método setter chamado `setItemName:`. (Observe que essas declarações não aparecerão no seu arquivo. Elas são declaradas pelo compilador em segundo plano.) Portanto, o restante do código no aplicativo pode funcionar como antes.

A declaração dessas propriedades também cuida das implementações dos métodos acessores. No `BNRItem.m`, exclua as implementações dos métodos acessores.

```
- (void)setItemName:(NSString *)str
{
    _itemName = str;
}
-(NSString *)itemName
{
    return _itemName;
}
-(void)setSerialNumber:(NSString *)str
{
    _serialNumber = str;
}
-(NSString *)serialNumber
{
    return _serialNumber;
}
-(void)setValueInDollars:(int)p
{
    _valueInDollars = p;
}
-(int)valueInDollars
{
    return _valueInDollars;
}
```

```

- (NSDate *)dateCreated
{
    return _dateCreated;
}
-(void)setContainedItem:(BNRItem *)item
{
    _containedItem = item;

    // When given an item to contain, the contained
    // item will be given a pointer to its container
    item.container = self;
}
-(BNRItem *)containedItem
{
    return _containedItem;
}

-(void)setContainer:(BNRItem *)item
{
    _container = item;
}
-(BNRItem *)container
{
    return _container;
}

```

Você deve estar se perguntando sobre a implementação de `setContainedItem:` que você acabou de excluir. Esse método setter fez mais do que apenas definir a variável de instância `_containedItem`. Ele também definiu a variável de instância `_container` do item que foi passado. Para replicar essa funcionalidade, você logo vai escrever um setter personalizado para a propriedade `containedItem`. Mas primeiro, vamos falar sobre alguns princípios básicos de propriedades.

## Atributos de propriedades

Uma propriedade tem uma série de atributos que lhe permitem modificar o comportamento dos métodos acessores e a variável de instância que ela cria. Os atributos são declarados entre parênteses, depois da diretiva `@property`. Veja um exemplo:

```
@property (nonatomic, readwrite, strong) NSString *itemName;
```

Cada atributo tem um conjunto de valores possíveis, dos quais um é o valor padrão que não precisa ser declarado explicitamente.

### Atributo multisegmentado

O atributo multisegmentado de uma propriedade tem dois valores: `nonatomic` (não atômico) ou `atomic` (atômico). (A multisegmentação está fora do escopo deste livro, mas você ainda precisa saber os valores para esse atributo.) A maioria dos programadores de iOS normalmente usa a opção `nonatomic`: nós usamos na Big Nerd Ranch, e a Apple também usa. Neste livro, você usará o atributo `nonatomic` para todas as propriedades.

Infelizmente, o valor padrão para esse atributo é `atomic`, então você precisa especificar que deseja que suas propriedades sejam `nonatomic`.

No `BNRItem.h`, modifique todas as suas propriedades para que sejam `nonatomic`.

```
@interface BNRItem : NSObject  
+ (instancetype)randomItem;  
- (instancetype)initWithItemName:(NSString *)name  
    valueInDollars:(int)value  
    serialNumber:(NSString *)sNumber;  
- (instancetype)initWithItemName:(NSString *)name;  
  
@property (nonatomic) BNRItem *containedItem;  
@property (nonatomic) BNRItem *container;  
  
@property (nonatomic) NSString *itemName;  
@property (nonatomic) NSString *serialNumber;  
@property (nonatomic) int valueInDollars;  
@property (nonatomic) NSDate *dateCreated;  
@end
```

## Atributo de leitura/gravação

O valor de leitura e gravação do atributo, `readwrite` (leitura/gravação) ou `readonly` (somente leitura), diz ao compilador se ele precisa implementar um método setter para a propriedade. Uma propriedade `readwrite` implementa os dois métodos, setter e getter. Uma propriedade `readonly` implementa apenas um método getter. A opção padrão para este atributo é `readwrite` (leitura/gravação). Isso é o que você quer para todas as propriedades de `BNRItem`, com exceção de `dateCreated`, que deve ser `readonly` (somente leitura).

No `BNRItem.h`, declare `dateCreated` como uma propriedade `readonly` para que o método setter seja gerado para essa variável de instância.

```
@property (nonatomic, readonly) NSDate *dateCreated;
```

## Atributo de gerenciamento de memória

Os valores do atributo de gerenciamento de memória são: referência `strong` (forte), `weak` (fraca), `copy` (cópia) e `unsafe_unretained` (insegura não retida) Esse atributo descreve o tipo de referência que o objeto com a variável de instância tem para o objeto para o qual a variável está apontando.

Para propriedades que não apontam para objetos (como `valueInDollars` do tipo `int`), não há necessidade de gerenciamento de memória e a única opção é `unsafe_unretained` (insegura não retida). Isso é atribuição direta. Você também pode ver o valor `assign` (atribuir) em alguns lugares, que era o termo usado antes da ARC.

(A parte “insegura” de `unsafe_unretained` pode enganar quando lidamos com propriedades do tipo não objeto. A confusão vem do contraste de referências inseguras não retidas com referências fracas. Diferente de uma referência fraca, a referência insegura não retida não é definida para `nil` automaticamente, quando o objeto para o qual aponta é destruído. Isso é inseguro porque você pode acabar tendo ponteiros pendentes. Entretanto, a questão de ponteiros pendentes é irrelevante quando lidamos com propriedades do tipo não objeto.)

Como a única opção, `unsafe_unretained` também é o valor padrão para as propriedades do tipo não objeto, assim você pode deixar a propriedade `valueInDollars` como está.

Para propriedades que gerenciam um ponteiro para um objeto em Objective-C, as quatro opções são possíveis. O padrão é a opção `strong`. Contudo, os programadores de Objective-C tendem a declarar esse atributo explicitamente. (Um dos motivos é que o valor padrão mudou nos últimos anos, e isso poderia acontecer novamente.)

No `BNRItem.m`, defina o atributo de gerenciamento de memória como `strong` para as propriedades `containedItem` e `dateCreated`, e `weak` para a propriedade `container`.

```
@property (nonatomic, strong) BNRItem *containedItem;  
@property (nonatomic, weak) BNRItem *container;  
  
@property (nonatomic) NSString *itemName;  
@property (nonatomic) NSString *serialNumber;  
@property (nonatomic) int valueInDollars;  
@property (nonatomic, readonly, strong) NSDate *dateCreated;
```

Definir a propriedade `container` para `weak` evita o ciclo de referência forte que você causou e corrigiu anteriormente.

E quanto às propriedades `itemName` e `serialNumber`? Elas apontam para instâncias de **NSString**. Quando uma propriedade aponta para a instância de uma classe que tem uma subclasse mutável (como **NSString/NSMutableString** ou **NSArray/NSMutableArray**), você deve definir seu atributo de gerenciamento de memória para `copy`.

No `BNRItem.m`, defina o atributo de gerenciamento de memória para `itemName`, e `serialNumber` como `copy`.

```
@property (nonatomic, strong) BNRItem *containedItem;
@property (nonatomic, weak) BNRItem *container;

@property (nonatomic, copy) NSString *itemName;
@property (nonatomic, copy) NSString *serialNumber;
@property (nonatomic) int valueInDollars;
@property (nonatomic, readonly, strong) NSDate *dateCreated;
```

O método setter gerado para `itemName` será o seguinte:

```
- (void)setitemName:(NSString *)itemName
{
    _itemName = [itemName copy];
}
```

Em vez de definir `_itemName` para apontar para o objeto de entrada, esse setter envia a mensagem `copy` para aquele objeto. O método `copy` retorna um objeto imutável **NSString** que tem os mesmos valores que a string original, e `_itemName` é definida para apontar para uma nova string.

Por que é mais seguro fazer isso para **NSString**? É mais seguro fazer uma cópia do objeto em vez de correr o risco de apontar para um objeto possivelmente mutável, que pode ter outros proprietários que, por sua vez, poderiam mudar o objeto sem o seu conhecimento.

Por exemplo, imagine se um item fosse inicializado de forma que `itemName` apontasse para uma **NSMutableString**.

```
NSMutableString *mutableString = [[NSMutableString alloc] init];
BNRItem *item = [[BNRItem alloc] initWithitemName:mutableString
                                         valueInDollars:5
                                         serialNumber:@"4F2W7"]];
```

Esse código é válido porque uma instância de **NSMutableString** também é uma instância de sua superclasse, **NSString**. O problema é que a string apontada por `mutableString` pode ser alterada sem o conhecimento do item que também aponta para ela.

Em seu aplicativo, você não mudará essa string, a menos que queira. No entanto, quando você escreve classes para que outras pessoas as usem, você não pode controlar como elas serão usadas, e tem de programar de forma defensiva.

Neste caso, a defesa é declarar `itemName` com o atributo `copy`.

Em termos de propriedade, `copy` fornece uma referência forte para o objeto apontado. A string original não é modificada de nenhuma maneira: não ganha nem perde um proprietário, e nenhum dos dados é alterado.

Embora seja prudente fazer uma cópia de um objeto mutável, é perda de tempo fazer uma cópia de um objeto imutável. Um objeto imutável não pode ser alterado, portanto, o tipo de problema descrito acima não pode ocorrer. Para evitar a cópia desnecessária, classes imutáveis implementam `copy` para retornar silenciosamente um ponteiro para o objeto original e imutável.

## Acessores personalizados com propriedades

Por padrão, os métodos acessores que uma propriedade implementa são bastante simples e têm a seguinte aparência:

```
- (void)setContainedItem:(BNRItem *)item
{
    _containedItem = item;
}
(BNRItem *)containedItem
{
    return _containedItem;
}
```

Para a maioria das propriedades, é exatamente isso o que você deseja. No entanto, para a propriedade `containedItem`, o método setter padrão não é suficiente. A implementação de `setContainedItem:` precisa de outra etapa: ela também deve definir a propriedade `container` do item que está sendo contido.

Você pode substituir o setter padrão implementando-o você mesmo no arquivo de implementação.

No `BNRItem.m`, adicione de volta uma implementação para `setContainedItem:`.

```
- (void)setContainedItem:(BNRItem *)containedItem
{
    _containedItem = containedItem;
    self.containedItem.container = self;
}
```

Quando o compilador vir que você implementou `setContainedItem:`, ele não criará um setter padrão para `containedItem`. Ele ainda criará o método getter, `containedItem`.

Observe que se você implementar tanto um setter personalizado quanto um getter personalizado (ou apenas um getter personalizado em uma propriedade somente leitura), o compilador não criará uma variável de instância para sua propriedade. Se precisar de uma, você mesmo deve declará-la.

Observe a moral: às vezes os acessores padrão não fazem o que você precisa, e você mesmo terá de implementá-los.

Agora você pode compilar e executar o aplicativo. A `BNRItem` aprendiz funciona da mesma maneira.

## Para os mais curiosos: síntese de propriedades

Ao explicar propriedades neste capítulo, observamos que uma propriedade gera automaticamente a implementação para os métodos acessores, declara e cria uma variável de instância. Embora seja verdade, nós omitimos o fato que este comportamento é apenas o padrão, e você tem outras opções.

Declarar uma propriedade em uma interface de classe declara apenas os métodos acessores na interface de classe. Para que uma propriedade gere automaticamente uma variável de instância e as implementações de seus métodos, ela precisa ser *sintetizada*, implícita ou explicitamente. As propriedades são sintetizadas implicitamente por padrão. Uma propriedade é sintetizada explicitamente usando a diretiva `@synthesize` (sintetizar) no arquivo de implementação:

```
@implementation Person

// Generates the code for -setAge: and -age,
// and creates the instance variable _age
@synthesize age = _age;

// Other methods go here

@end
```

É assim que as propriedades são sintetizadas automaticamente. O primeiro atributo, `age` (idade), diz: “crie métodos chamados `age` e `setAge:`”, e o segundo atributo (`_age`) diz: “a variável de instância que dá suporte a esses métodos deve ser `_age`”.

Opcionalmente, você pode deixar o nome da variável de fora, o que vai criar uma variável de apoio com o mesmo nome dos métodos acessores.

```
@synthesize age;
// Is the same as:
@synthesize age = age;
```

Há casos em que você não quer uma variável de instância para dar suporte a uma propriedade e, consequentemente, não quer que uma propriedade gere as implementações dos métodos acessores automaticamente. Considere uma classe `Person` com três propriedades: `spouse`, `lastName` e `lastNameOfSpouse`:

```
@interface Person : NSObject
@property (nonatomic, strong) Person *spouse;
@property (nonatomic, copy) NSString *lastName;
@property (nonatomic, copy) NSString *lastNameOfSpouse;
@end
```

Neste exemplo um tanto forçado, faz sentido que as duas propriedades, `spouse` e `lastName`, sejam apoiadas por uma variável de instância. Afinal de contas, essas são informações que cada **Person** precisa ter. No entanto, não faz sentido se ater ao sobrenome do cônjuge como uma variável de instância. Uma **Person** pode simplesmente pedir o `lastName` de sua `spouse`, assim, armazenar essa informação em todas as instâncias de **Person** é redundante e, portanto, suscetível a erros. Em vez disso, a classe **Person** implementaria os métodos `getter` e `setter` para a propriedade `lastNameOfSpouse` da seguinte forma:

```
@implementation Person
- (void)setLastNameOfSpouse:(NSString *)lastNameOfSpouse
{
    self.spouse.lastName = lastNameOfSpouse;
}

- (NSString *)lastNameOfSpouse
{
    return self.spouse.lastName;
}
@end
```

Neste caso, como você implementou ambos os acessores, o compilador não sintetizará uma variável de instância para `lastNameOfSpouse` automaticamente. Que é exatamente o que você esperaria.

## Para os mais curiosos: pool de liberação automática e história da ARC

Antes da contagem de referência automática (ARC) ser adicionada à linguagem Objective-C, tínhamos a *contagem de referência manual*. Com a contagem de referência manual, mudanças de propriedade aconteciam apenas quando você enviava uma mensagem explícita para um objeto.

```
[anObject release]; // anObject loses an owner
[anObject retain]; // anObject gains an owner
```

Era um problema: esquecer-se de enviar `release` para um objeto, antes de definir um ponteiro para apontar para alguma outra coisa, criaria uma perda de memória. Enviar `release` para um objeto se não tivesse enviado `retain` antes, era uma desalocação prematura. Muito tempo era perdido fazendo a depuração desses problemas, que poderiam se tornar bastante complexos em grandes projetos.

Durante os dias sombrios da contagem de referência manual, a Apple estava participando de um projeto de código-fonte aberto, conhecido como analisador estático Clang, e fazendo sua integração para o Xcode. Você verá mais sobre o analisador estático no Chapter 14, mas o essencial é que ele poderia analisar códigos e dizer se você estivesse fazendo alguma besteira. Duas das besteiras que ele poderia detectar eram perda de memória e desalocação prematura. Programadores espertos fariam a execução do código através do analisador estático para detectar estes problemas e, em seguida, escreveriam o código necessário para corrigi-los.

Com o tempo, o analisador estático ficou tão bom que a Apple pensou: “Por que não deixar que o analisador estático simplesmente insira todas as mensagens de retenção e liberação?” Assim nascia a ARC. As pessoas se alegraram nas ruas e os problemas de gerenciamento de memória viraram coisa do passado.

Outra coisa que os programadores tinham que entender nos tempos de contagem de referência manual era o *pool de liberação automática*. Quando um objeto recebia a mensagem `autorelease`, o pool de liberação automática obteria a propriedade de um objeto temporariamente, para que pudesse ser retornado a partir do método que o criou, sem sobrecarregar o criador ou o destinatário com as responsabilidades de propriedade. Isso era essencial para os métodos de conveniência, que criavam uma nova instância de algum objeto e o retornavam:

```
+ (BNRItem *)someItem
{
    BNRItem *item = [[[BNRItem alloc] init] autorelease];
    return item;
}
```

Como você tinha de enviar a mensagem `release` para que um objeto renunciasse à propriedade, o autor da chamada desse método tinha de entender suas responsabilidades de propriedade. Mas era fácil se confundir.

```
BNRItem *item = [BNRItem someItem]; // I guess I own this now?  
NSString *string = [item itemName]; // Well, if I own that, do I own this?
```

Sendo assim, os objetos criados por métodos que não **alloc** e **copy** receberiam **autorelease** antes de serem retornados e o destinatário do objeto assumiria a propriedade, conforme necessário, ou simplesmente deixaria que fosse destruído quando o pool de liberação automática ficasse vazio.

Com a ARC, isso é feito automaticamente (e, algumas vezes, otimizado completamente). Um pool de liberação automática é criado pela diretiva `@autoreleasepool`, seguida de chaves `{}`. Dentro das chaves, qualquer objeto recentemente instanciado, retornado de um método que não tenha **alloc** ou **copy** no nome, é colocado no pool de liberação automática. Quando as chaves são fechadas, qualquer objeto no pool perde um proprietário.

```
@autoreleasepool {  
    // Get a BNRItem back from a method that created it,  
    // method does not say alloc/copy  
    BNRItem *item = [BNRItem someItem];  
} // Pool is drained, item loses an owner and is destroyed
```

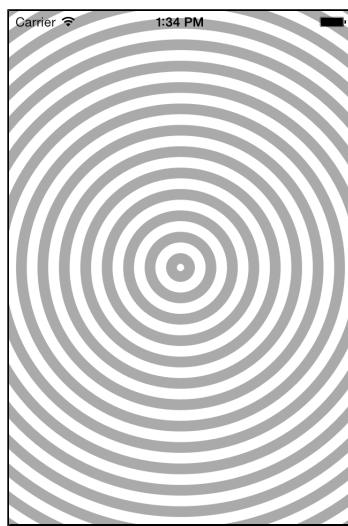
Aplicativos iOS criam automaticamente um pool de liberação automática para você, e você realmente não tem de se preocupar com isso. Mas não é legal saber para que serve aquele `@autoreleasepool`?

# 4

## Visões e a hierarquia de visões

Neste capítulo, você aprenderá sobre visões e a hierarquia de visões. Em especial, você vai aprender a construir um aplicativo chamado Hypnosister que projeta um conjunto de círculos concêntricos que ocupa toda a tela.

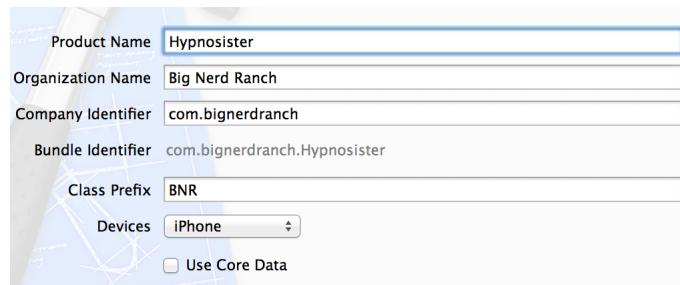
Figure 4.1 Hypnosister



No Xcode, selecione File → New → Project... (ou use o atalho no teclado Command-Shift-N). Na seção iOS, selecione Application, escolha o template Empty Application e clique em Next.

Digite Hypnosister como o nome do produto e BNR como o prefixo de classe, conforme mostrado na Figure 4.2. Certifique-se de que iPhone esteja selecionado no menu suspenso Devices e de que a caixa Use Core Data esteja desmarcada.

Figure 4.2 Configuração do Hypnosister



O Hypnosister não terá nenhuma interação do usuário, para que você possa se concentrar na maneira como as visões são projetadas na tela. Vamos começar com uma pequena teoria de visões e da hierarquia de visões.

### Fundamentos da visão

- Uma visão é uma instância de **UIView** ou de uma de suas subclasses.

- Uma visão sabe como se projetar.
- Uma visão lida com eventos, como toques.
- Uma visão existe dentro de uma hierarquia de visões. A raiz dessa hierarquia é a janela do aplicativo.

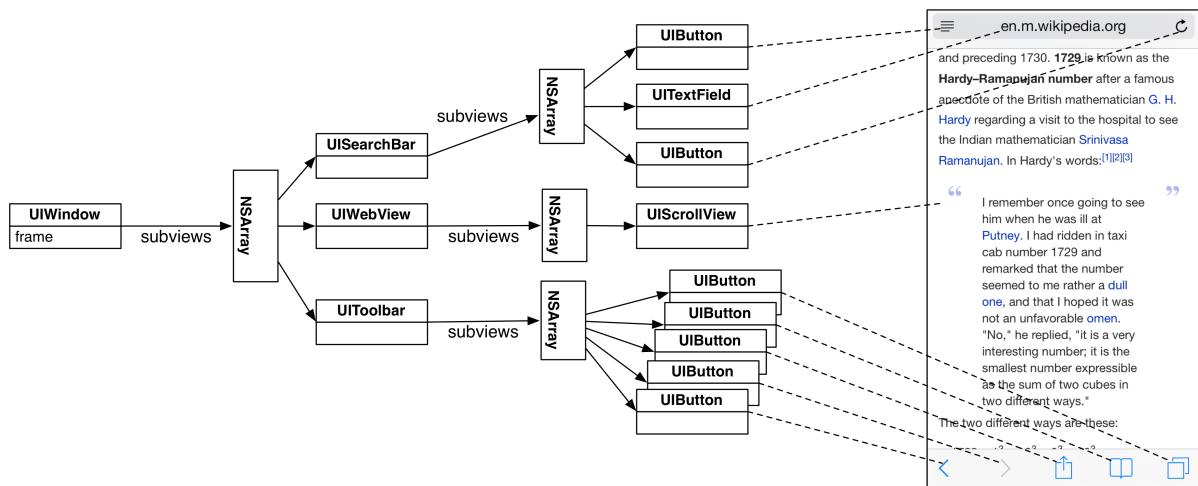
No Chapter 1, você criou quatro visões para o aplicativo Quiz: duas instâncias de **UIButton** e duas instâncias de **UILabel**. Você criou e configurou essas visões no Interface Builder, mas também pode criar visões programaticamente. No Hypnosister, você criará visões programaticamente.

## A hierarquia de visões

Cada aplicativo tem exatamente uma instância de **UIWindow** que serve como contêiner para todas as visões do aplicativo. A janela é criada quando o aplicativo é iniciado. Depois que a janela é criada, você pode adicionar outras visões a ela.

Quando uma visão é adicionada a uma janela, ela é chamada de *subvisão* da janela. As visões que são subvisões da janela também podem ter subvisões, e o resultado é uma hierarquia de objetos do tipo visão com a janela em sua raiz.

Figure 4.3 Um exemplo de hierarquia de visões e da interface que ela cria

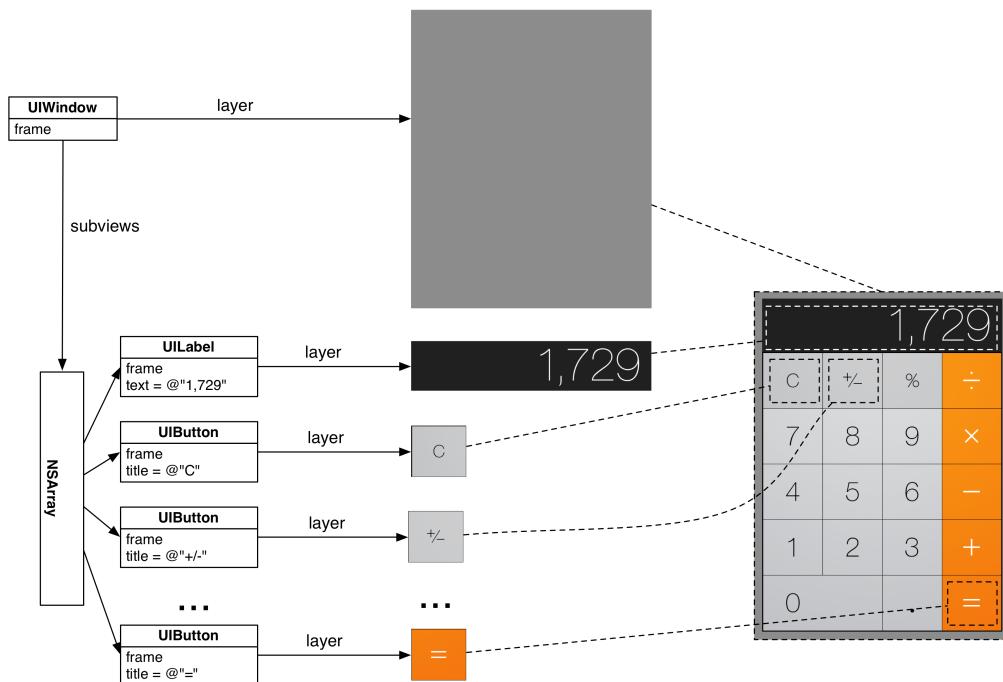


Uma vez que a hierarquia tenha sido criada, ela será projetada na tela. Esse processo pode ser dividido em duas etapas:

- Cada visão na hierarquia, incluindo a janela, projeta a si própria. Ela renderiza a si própria para sua *camada*, que é uma instância de **CALayer**. (Você pode pensar na camada de uma visão como se fosse uma imagem bitmap.)
- As camadas de todas as visões são compostas juntas na tela.

A Figure 4.4 mostra outro exemplo de hierarquia de visões e as duas etapas de projeção.

Figure 4.4 Visões renderizam a si mesmas e, em seguida, são compostas juntas



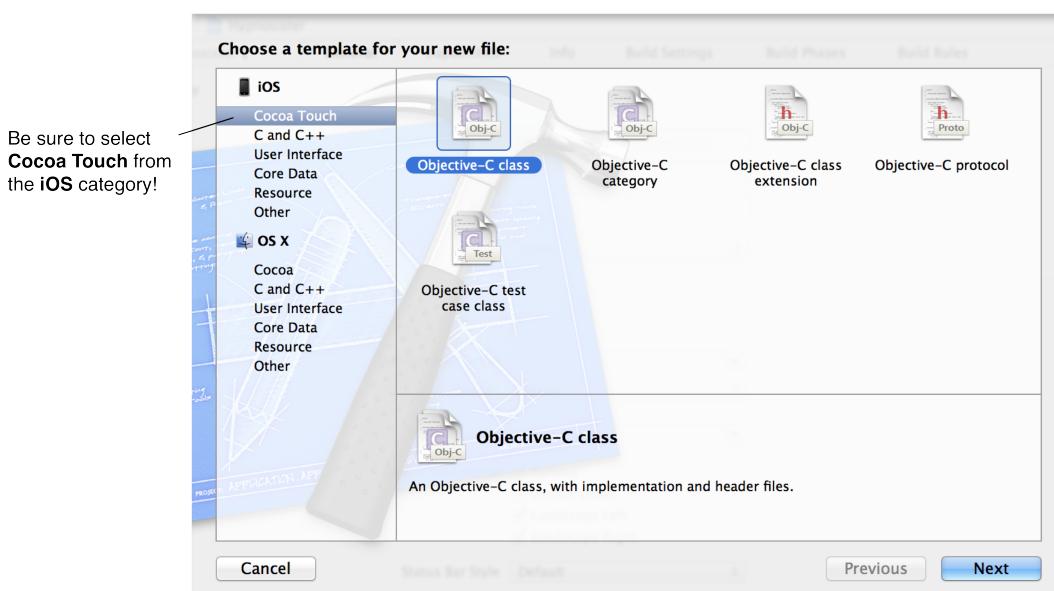
Classes como **UIButton** e **UILabel** já sabem como renderizar a si próprias para suas camadas. Por exemplo, no Quiz, você criou instâncias de **UILabel** e disse a elas que texto exibir, mas não precisou dizer como desenhar o texto. Os desenvolvedores da Apple cuidaram disso.

A Apple, no entanto, não fornece uma classe cujas instâncias saibam como desenhar círculos concêntricos. Assim, para o Hypnosister, você vai criar sua própria subclasse **UIView** e construir o código de projeção personalizado.

## Criação da subclasse UIView

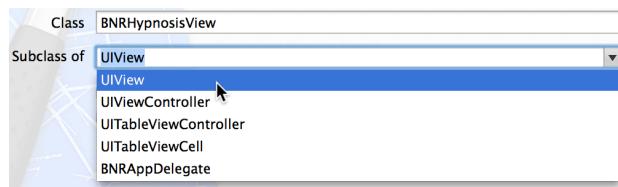
Para criar uma subclasse **UIView**, selecione File → New → File... (ou pressione Command-N). Na seção iOS, selecione Cocoa Touch e depois Objective-C class (Figure 4.5).

Figure 4.5 Criação de uma nova classe



Clique em Next. No painel seguinte, nomeie a classe como **BNRHypnosisView** e selecione **UIView** como a superclasse (Figure 4.6).

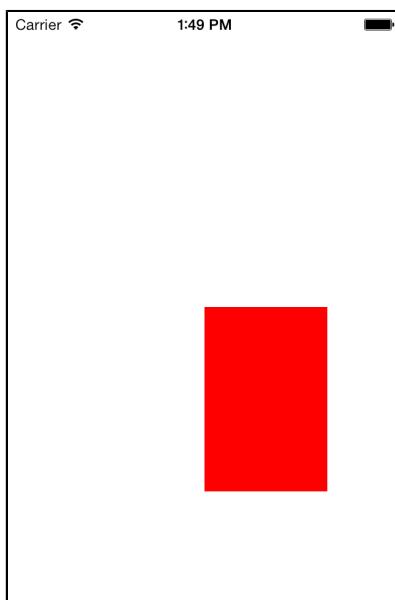
Figure 4.6 Escolha de **UIView** como superclasse



Clique em Next. Certifique-se de que Hypnosister esteja marcado ao lado de Targets e, em seguida, clique em Create.

Antes de escrever o código de projeção de círculos concêntricos para **BNRHypnosisView**, vamos focar em como criar programaticamente uma visão e colocá-la na tela. Para manter a simplicidade, nesta primeira parte, uma instância da visão **BNRHypnosisView** não projetará círculos concêntricos. Em vez disso, ela projetará um retângulo com um fundo vermelho.

Figure 4.7 Versão inicial de **BNRHypnosisView**



## Visões e frames

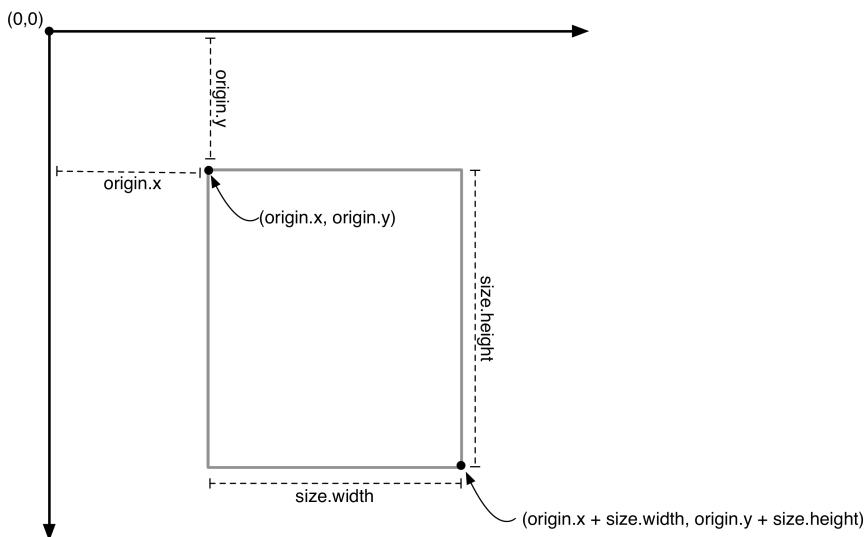
Abra o `BNRHypnosisView.m`. O template da subclasse **UIView** forneceu dois stubs de método para você. O primeiro é `initWithFrame:`, o inicializador designado para **UIView**. Esse método recebe um argumento, um `CGRect`, que se tornará o `frame` da visão, uma propriedade na **UIView**.

```
@property (nonatomic) CGRect frame;
```

O `frame` de uma visão especifica o tamanho da visão e sua posição em relação à sua supervisão. Como o tamanho de uma visão é sempre especificado por seu `frame`, uma visão é sempre um retângulo.

Um `CGRect` contém os membros `origin` e `size`. A variável `origin` é uma estrutura em C do tipo `CGPoint` e contém dois membros `float`: `x` e `y`. A variável `size` é uma estrutura em C do tipo `CGSize` e contém dois membros `float`: `width` e `height` (Figure 4.8).

Figure 4.8 CGRect



Abra o `BNRAppDelegate.m`. No topo desse arquivo, importe o arquivo de cabeçalho da `BNRHypnosisView`.

```
#import "BNRAppDelegate.h"
#import "BNRHypnosisView.h"

@implementation BNRAppDelegate
```

No `BNRAppDelegate.m`, encontre a implementação de `application:didFinishLaunchingWithOptions:` do template. Após a linha que cria a janela, crie um `CGRect` que será o frame de uma `BNRHypnosisView`. Em seguida, crie uma instância de `BNRHypnosisView` e configure sua propriedade `backgroundColor` como vermelho. Por fim, adicione a `BNRHypnosisView` como uma subvisão da janela para torná-la parte da hierarquia de visão.

```
- (BOOL)application:(UIApplication *)application
    didFinishLaunchingWithOptions:(NSDictionary *)launchOptions
{
    self.window = [[UIWindow alloc] initWithFrame:[[UIScreen mainScreen] bounds]];
    CGRect firstFrame = CGRectMake(160, 240, 100, 150);

    BNRHypnosisView *firstView = [[BNRHypnosisView alloc] initWithFrame:firstFrame];
    firstView.backgroundColor = [UIColor redColor];

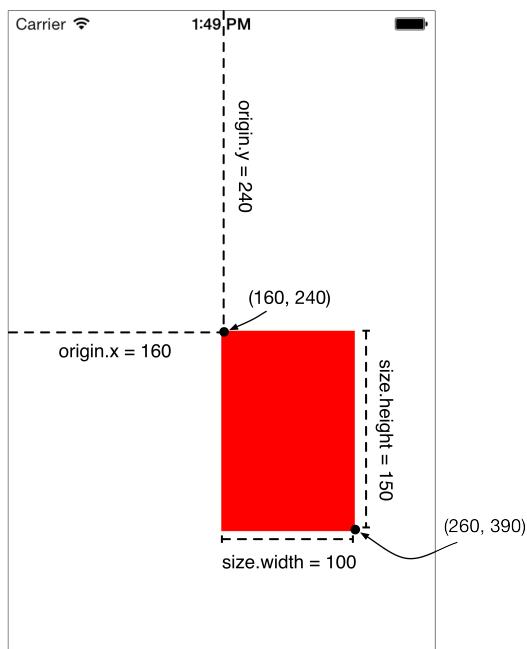
    [self.window addSubview:firstView];

    self.window.backgroundColor = [UIColor whiteColor];
    [self.window makeKeyAndVisible];
    return YES;
}
```

Uma estrutura não é um objeto Objective-C, e, por isso, você não pode enviar mensagens para um `CGRect`. Para criar um, você usou `CGRectMake()` e passou os valores para `origin.x`, `origin.y`, `size.width` e `size.height`. Um `CGRect` é pequeno em comparação com a maioria dos objetos; então, em vez de passar um ponteiro para ele, você simplesmente passa a estrutura inteira. Assim, `initWithFrame:` espera um `CGRect`, não um `CGRect *`.

Para configurar `backgroundColor`, você usou o método de classe `UIColor redColor`. Esse é um método de conveniência; ele aloca e inicializa uma instância de `UIColor`, que é configurada para ser vermelha. Há diversos métodos de conveniência `UIColor` para cores comuns, como `blueColor`, `blackColor` e `clearColor`.

Compile e execute o aplicativo. O retângulo vermelho é a instância de `BNRHypnosisView`. Como a `origin` do frame da `BNRHypnosisView` é `(160, 240)`, seu canto superior esquerdo está 160 pontos à direita e 240 pontos abaixo do canto superior esquerdo da janela (sua supervisão). A visão se estende 100 pontos para a direita e 150 pontos para baixo a partir de sua `origin`, de acordo com o tamanho, `size`, de seu frame.

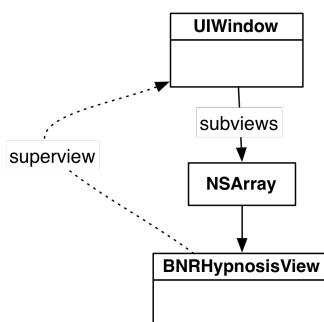
Figure 4.9 Hypnosister com uma **BNRHypnosisView**

Observe que esses valores estão em pontos, não em pixels. Se os valores estivessem em pixels, eles não seriam consistentes por telas com diferentes resoluções (ou seja, Retina vs. não-Retina). Um único ponto é uma unidade relativa de uma medida; ele será um número diferente de pixels dependendo de quantos pixels houver na tela. Tamanhos, posições, linhas e curvas são sempre descritos em pontos para permitir diferenças na resolução da tela.

Em uma tela Retina, um pixel tem meio ponto de altura e meio ponto de largura por padrão. Em uma tela que não seja Retina, um pixel tem um ponto de altura e um ponto de largura por padrão. Ao imprimir em papel, uma polegada tem 72 pontos de comprimentos.

No console do Xcode, observe o comentário informando que “Application windows are expected to have a root view controller at the end of application launch.” (Espera-se que as janelas do aplicativo tenham um controlador de visão raiz ao final da abertura do aplicativo). Um controlador de visão é um objeto que controla algum conjunto da hierarquia de visão de um aplicativo, e a maioria dos aplicativos do iOS tem um ou mais controladores de visão. O Hypnosister, porém, é simples o suficiente para não precisar de um controlador de visão, e, por isso, você pode ignorar esse comentário. Você aprenderá sobre controladores de visão no Chapter 6.

Examine a hierarquia de visão que você criou:

Figure 4.10 **UIWindow** tem uma subvisão - uma **BNRHypnosisView**

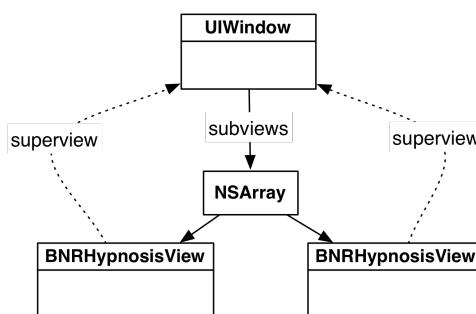
Toda instância de **UIView** tem uma propriedade **superview**. Quando você adiciona uma visão como subvisão de outra visão, a relação inversa é automaticamente estabelecida. Nesse caso, a **superview** da **BNRHypnosisView** é a **UIWindow**. (Para evitar um ciclo de referência forte, a propriedade **superview** é uma referência fraca.)

Vamos fazer experiências com a sua hierarquia de visão. No `BNRAppDelegate.m`, crie outra instância de `BNRHypnosisView` com um frame e uma cor de fundo diferentes.

```
...
[self.window addSubview:firstView];
CGRect secondFrame = CGRectMake(20, 30, 50, 50);
BNRHypnosisView *secondView = [[BNRHypnosisView alloc] initWithFrame:secondFrame];
secondView.backgroundColor = [UIColor blueColor];
[self.window addSubview:secondView];
self.window.backgroundColor = [UIColor whiteColor];
...
```

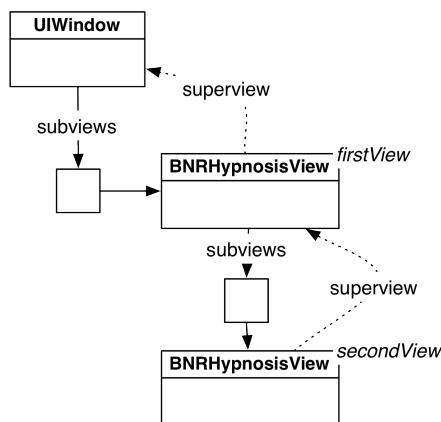
Compile e execute novamente. Além do retângulo vermelho, você verá um quadrado azul perto do canto superior esquerdo da janela. A Figure 4.11 mostra a hierarquia de visão atualizada.

Figure 4.11 `UIWindow` tem duas subvisões como irmãs



Uma hierarquia de visão pode ter uma profundidade maior que dois níveis. Vamos fazer isso adicionando a segunda instância de `BNRHypnosisView` como uma subvisão da primeira instância de `BNRHypnosisView` em vez da janela:

Figure 4.12 Uma `BNRHypnosisView` como subvisão da outra

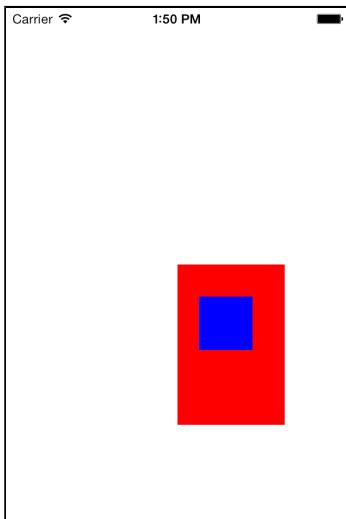


No `BNRAppDelegate.m`, faça esta alteração.

```
...
BNRHypnosisView *secondView = [[BNRHypnosisView alloc] initWithFrame:secondFrame];
secondView.backgroundColor = [UIColor blueColor];
[self.window addSubview:secondView];
[firstView addSubview:secondView];
...
```

Compile e execute o aplicativo. Observe que a posição da `secondView` na tela mudou. O `frame` de uma visão é relativo a sua supervisão, e, por isso, o canto superior esquerdo da `secondView` agora está inserido a (20, 30) pontos do canto superior esquerdo da `firstView`.

Figure 4.13 Hypnosister com nova hierarquia



(Se a instância azul de `BNRHypnosisView` parecer menor do que parecia anteriormente, é apenas uma ilusão de ótica. Seu tamanho não se alterou.)

Agora que você já tem alguma experiência com a hierarquia de visão, remova a segunda instância de `BNRHypnosisView` antes de continuar.

```
...
[self.window addSubview:firstView];
CGRect secondFrame = CGRectMake(20, 30, 50, 50);
BNRHypnosisView *secondView = [[BNRHypnosisView alloc] initWithFrame:secondFrame];
secondView.backgroundColor = [UIColor blueColor];
[view addSubview:secondView];
self.window.backgroundColor = [UIColor whiteColor];
...
```

## Projeções personalizadas em `drawRect:`

Até agora, você criou a subclasse `UIView`, criou instâncias da subclasse, inseriu-as na hierarquia de visão e especificou seus frames e `backgroundColors`. Nesta seção, você escreverá o código de projeção personalizado para `BNRHypnosisView` em seu método `drawRect:`.

O método `drawRect:` é a etapa de renderização na qual uma visão se projeta sobre sua camada. Subclasses `UIView` sobrescrevem o `drawRect:` para realizar a projeção personalizada. Por exemplo, o método `drawRect:` de `UIButton` projeta texto azul-claro centralizado dentro de um retângulo.

A primeira coisa que você fará normalmente ao sobrescrever `drawRect:` será obter o retângulo de `bounds` da visão. A propriedade `bounds`, herdada de `UIView`, é o retângulo que define a área na qual a visão se projetará.

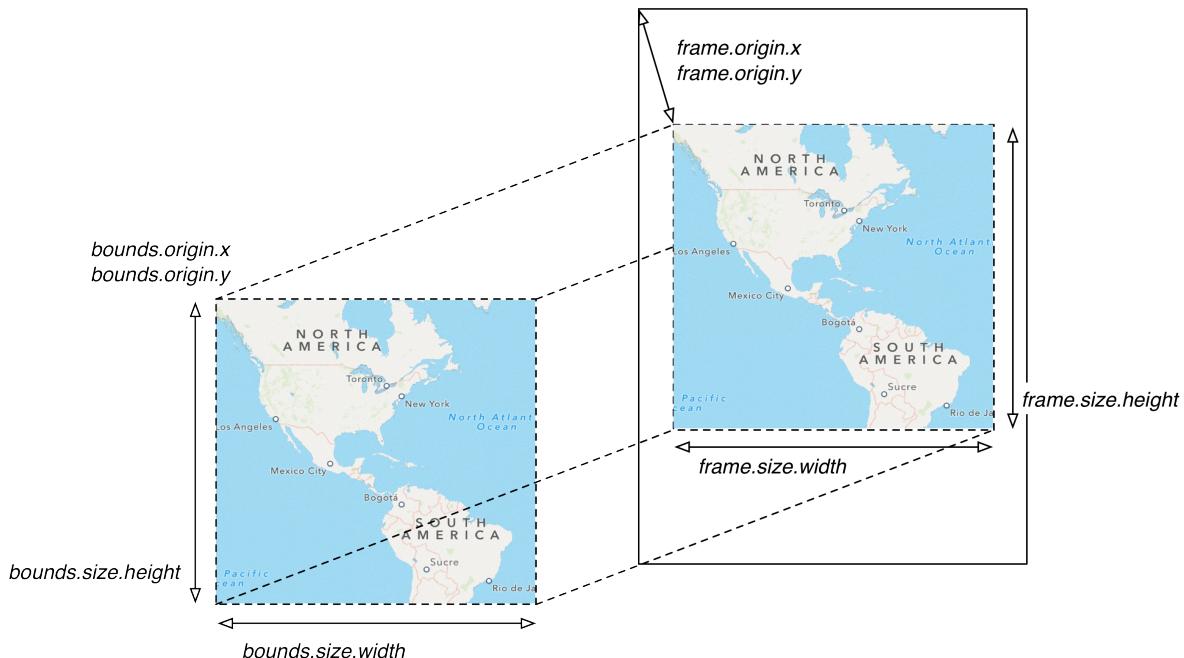
Cada visão tem um sistema de coordenadas que ela usa ao se projetar. `bounds` é o retângulo de uma visão em *seu próprio* sistema de coordenadas. `frame` é o mesmo retângulo no sistema de coordenadas da *sua supervisão*.

Você pode estar se perguntando: “Por que precisamos de outro retângulo se já temos `frame`?“

Os retângulos `frame` e `bounds` têm finalidades distintas. O retângulo `frame` de uma visão é usado durante a composição para dispor a camada da visão em relação ao resto da hierarquia de visão. O retângulo `bounds` é

usado durante a etapa de renderização para dispor a projeção detalhada dentro dos limites da camada da visão. (Figure 4.14).

Figure 4.14 bounds vs. frame



Você pode usar a propriedade `bounds` da janela para definir o frame para uma instância de tela cheia de `BNRHypnosisView`.

No `BNRAppDelegate.m`, atualize o frame da `firstView` para combinar com os bounds da janela.

```
- (BOOL)application:(UIApplication *)application
didFinishLaunchingWithOptions:(NSDictionary *)launchOptions
{
    self.window = [[UIWindow alloc] initWithFrame:[[UIScreen mainScreen] bounds]];
    // Override point for customization after application launch
    CGRect firstFrame = CGRectMake(160, 240, 100, 150);
    CGRect firstFrame = self.window.bounds;

    BNRHypnosisView *firstView = [[BNRHypnosisView alloc] initWithFrame:firstFrame];
    [self.window addSubview:firstView];

    self.window.backgroundColor = [UIColor whiteColor];
    [self.window makeKeyAndVisible];
    return YES;
}
```

Compile e execute o aplicativo, e você receberá uma visão em tamanho total com um fundo vermelho.

## Projeção de um único círculo

Você vai ser apresentado lentamente ao código de projeção projetando um único círculo – o maior que possa caber dentro dos limites (bounds) da visão.

No `BNRHypnosisView.m`, adicione código a `drawRect:`, que encontra o ponto central de bounds.

```
- (void)drawRect:(CGRect)rect
{
    CGRect bounds = self.bounds;

    // Figure out the center of the bounds rectangle
    CGPoint center;
    center.x = bounds.origin.x + bounds.size.width / 2.0;
    center.y = bounds.origin.y + bounds.size.height / 2.0;
}
```

Em seguida, configure o raio de seu círculo para que seja metade da menor das dimensões da visão. (Determinar a dimensão menor projetará o círculo direito em orientações retrato e paisagem.)

```
- (void)drawRect:(CGRect)rect
{
    CGRect bounds = self.bounds;

    // Figure out the center of the bounds rectangle
    CGPoint center;
    center.x = bounds.origin.x + bounds.size.width / 2.0;
    center.y = bounds.origin.y + bounds.size.height / 2.0;

    // The circle will be the largest that will fit in the view
    float radius = (MIN(bounds.size.width, bounds.size.height) / 2.0);

}
```

## UIBezierPath

O próximo passo é projetar o círculo usando a classe **UIBezierPath**. Instâncias dessa classe definem e projetam linhas e curvas que você pode usar para criar formas, como círculos.

Primeiro, crie uma instância de **UIBezierPath**.

```
- (void)drawRect:(CGRect)rect
{
    ...

    // The circle will be the largest that will fit in the view
    float radius = (MIN(bounds.size.width, bounds.size.height) / 2.0);

    UIBezierPath *path = [[UIBezierPath alloc] init];

}
```

O passo seguinte é definir o caminho que o objeto **UIBezierPath** deve seguir. Como você define um caminho em formato de círculo? O melhor lugar onde encontrar uma resposta para essa pergunta é a referência de classe **UIBezierPath** na documentação do desenvolvedor da Apple.

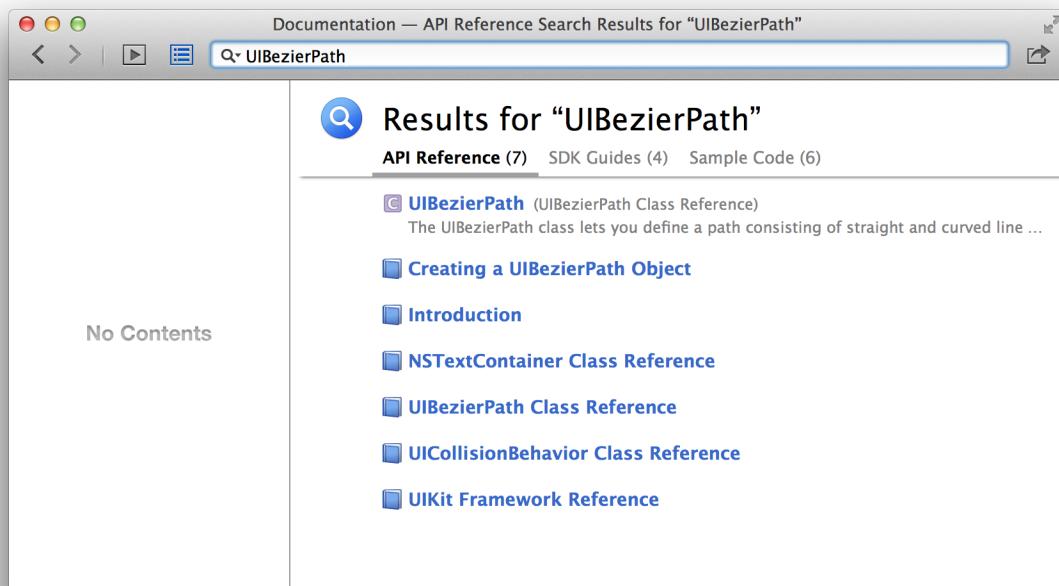
## Utilização da documentação do desenvolvedor

No menu do Xcode, selecione Help → Documentation and API Reference. Você também pode usar o atalho do teclado Option-Command-? (não deixe de também manter pressionada a tecla Shift, para receber o '?').

(Quando você acessar a documentação, o Xcode pode tentar obter a mais recente da Apple para você. Sua ID Apple e sua senha podem ser solicitadas.)

Quando o navegador da documentação abrir, procure por **UIBezierPath**. Você receberá vários resultados. Encontre e selecione **UIBezierPath Class Reference**.

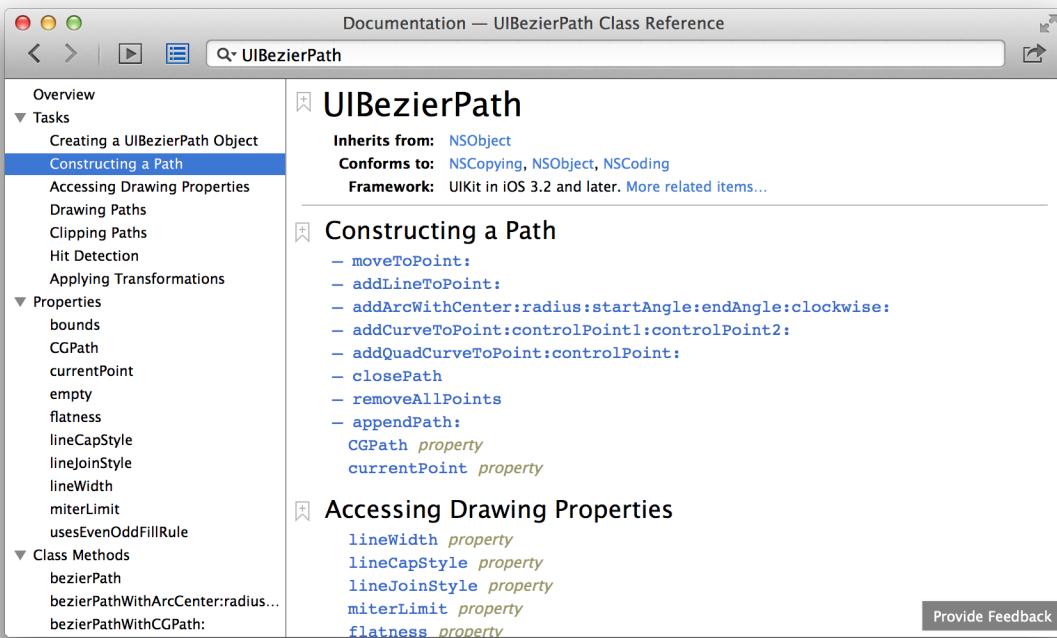
Figure 4.15 Resultados da documentação



Esta página é aberta para prover uma visão geral da classe, que é interessante, mas vamos nos focar na sua dúvida com relação ao caminho em formato circular. O lado esquerdo da referência é o índice. (Se você não vir um índice, selecione o ícone no canto superior esquerdo do navegador.)

No índice, encontre a seção **Tasks**. Esse é um bom lugar onde começar a busca por um método que faça algo específico. A primeira tarefa é **Creating a UIBezierPath Object**. Você já fez isso; então, dê uma olhada na segunda tarefa: **Constructing a Path**. Selecione essa tarefa e você verá uma lista de métodos **UIBezierPath** relevantes.

Figure 4.16 Métodos para a construção de um caminho



Um provável candidato para um caminho circular é o

**addArcWithCenter:radius:startAngle:endAngle:clockwise:**. Clique neste método para ver mais detalhes a respeito de seus parâmetros. Você já computou o centro e o raio. Os valores de ângulo inicial e final estão em radianos. Para projetar um círculo, você usará 0 para o ângulo inicial e M\_PI \* 2 para o ângulo final. (Se a sua trigonometria estiver enferrujada, você pode acreditar no que dizemos ou clicar no link Figure 1 dentro de Discussion da documentação desse método para ver um diagrama do círculo unitário.) Por fim, como você está projetando um círculo completo, o parâmetro de sentido horário não fará diferença. No entanto, ele é um parâmetro exigido, e, por isso, você precisará dar um valor a ele.

No BNRHypnosisView.m, envie uma mensagem à **UIBezierPath** que define seu caminho.

```
- (void)drawRect:(CGRect)rect
{
    CGRect bounds = self.bounds;

    // Figure out the center of the bounds rectangle
    CGPoint center;
    center.x = bounds.origin.x + bounds.size.width / 2.0;
    center.y = bounds.origin.y + bounds.size.height / 2.0;

    // The circle will be the largest that will fit in the view
    float radius = (MIN(bounds.size.width, bounds.size.height) / 2.0);

    UIBezierPath *path = [[UIBezierPath alloc] init];

    // Add an arc to the path at center, with radius of radius,
    // from 0 to 2*PI radians (a circle)
    [path addArcWithCenter:center
                      radius:radius
                 startAngle:0.0
                   endAngle:M_PI * 2.0
                  clockwise:YES];
}
```

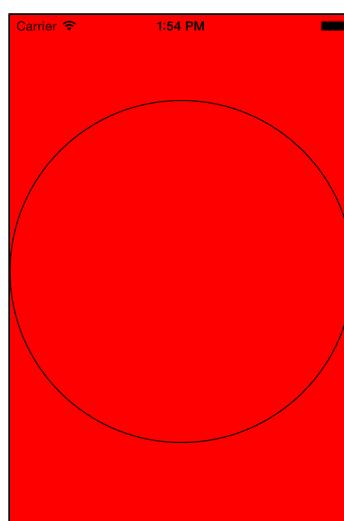
Você definiu um caminho, mas ainda não projetou nada. Voltando à referência de classe **UIBezier**, encontre e selecione a tarefa Drawing Paths. Dentre esses métodos, a melhor opção é **stroke**. (Os outros métodos preenchem toda a forma ou exigem um CGBlendMode do qual você não precisa.)

No `BNRHypnosisView.m`, envie uma mensagem à **UIBezierPath** que lhe diz para projetar.

```
- (void)drawRect:(CGRect)rect
{
    ...
    UIBezierPath *path = [[UIBezierPath alloc] init];
    // Add an arc to the path at center, with radius of radius,
    // from 0 to 2*PI radians (a circle)
    [path addArcWithCenter:center
                        radius:radius
                      startAngle:0.0
                     endAngle:M_PI * 2.0
                    clockwise:YES];
    // Draw the line!
    [path stroke];
}
```

Compile e execute o aplicativo, e você verá um fino contorno preto de um círculo que tem a largura da tela (ou a altura, se você estiver na orientação paisagem).

Figure 4.17 **BNRHypnosisView** com um único círculo



Com base no plano original para o Hypnosister, a linha que descreve seu círculo ainda não está correta. Ela deve ser mais larga e cinza-claro.

Para ver como resolver esses problemas, retorne à referência **UIBezierPath**. No índice, encontre a seção **Properties**. Uma dessas propriedades deve se destacar como sendo útil nesse caso – `lineWidth`. Selecione essa propriedade. Você verá que a `lineWidth` é do tipo `CGFloat` e que o padrão dela é 1.0.

Na `BNRHypnosisView.m`, transforme a largura da linha em 10 pontos.

```
- (void)drawRect:(CGRect)rect
{
    ...
    // Add an arc to the path at center, with radius of radius,
    // from 0 to 2*PI radians (a circle)
    [path addArcWithCenter:center
        radius:radius
        startAngle:0.0
        endAngle:M_PI * 2.0
        clockwise:YES];

    // Configure line width to 10 points
    path.lineWidth = 10;

    // Draw the line!
    [path stroke];
}
```

Compile e execute o aplicativo para confirmar se a linha está agora mais larga.

Não há nenhuma propriedade em **UIBezierPath** que lide com a cor da linha. Mas há uma pista na visão geral da classe. Use o índice para retornar à Overview. No quinto parágrafo (de acordo com o momento em que isto foi escrito), há um texto entre parênteses dizendo: “Você configura a pincelada e a cor de preenchimento usando a classe **UIColor**.”

A classe **UIColor** contém um link, e, por isso, você pode clicar nela para ser levado diretamente para a referência de classe **UIColor**. Na seção Tasks da **UIColor**, selecione Drawing Operations e navegue pelos métodos associados. Para os seus fins, você pode usar **set** ou **setStroke**. Você usará **setStroke** para deixar seu código mais óbvio às outras pessoas.

O método **setStroke** é um método de instância, e, por isso, você precisa de uma instância de **UIColor** à qual enviá-lo. Lembre-se que **UIColor** tem métodos de conveniência que retornam cores comuns. Você pode ver esses métodos listados na seção Class Methods da referência **UIColor**, incluindo um chamado **lightGrayColor**.

Agora, você já tem a informação de que precisa. No **BNRHypnosisView.m**, adicione código para criar uma instância de **UIColor** cinza-claro e envie a ela a mensagem **setStroke** para que, quando o caminho for projetado, ele seja projetado em cinza-claro.

```
- (void)drawRect:(CGRect)rect
{
    ...
    // Configure line width to 10 points
    path.lineWidth = 10;

    // Configure the drawing color to light gray
    [[UIColor lightGrayColor] setStroke];

    // Draw the line!
    [path stroke];
}
```

Compile e execute o aplicativo, e você verá um contorno cinza-claro mais largo de um círculo.

A esta altura, você já deve ter percebido que a **backgroundColor** de uma visão é projetada independentemente do que o **drawRect:** faça. Frequentemente, você definirá a **backgroundColor** de uma visão personalizada como transparente, ou “translúcida”, para que apenas os resultados de **drawRect:** apareçam.

No **BNRAppDelegate.m**, remova o código que configura a cor de fundo da visão.

```
BNRHypnosisView *firstView = [[BNRHypnosisView alloc] initWithFrame:firstFrame];
firstView.backgroundColor = [UIColor redColor];

[self.window addSubview:view];
```

Depois, no **BNRHypnosisView.m**, acrescente código a **initWithFrame:** para definir a cor de fundo de cada **BNRHypnosisView** como transparente.

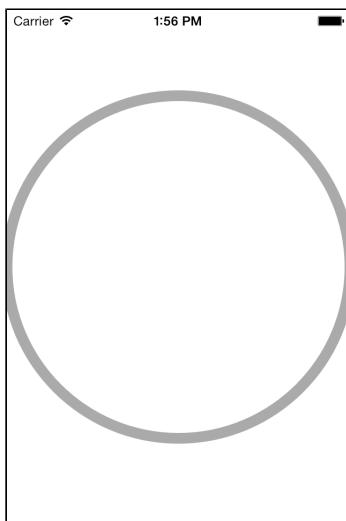
```

- (instancetype)initWithFrame:(CGRect)frame
{
    self = [super initWithFrame:frame];
    if (self) {
        // All BNRHypnosisViews start with a clear background color
        self.backgroundColor = [UIColor clearColor];
    }
    return self;
}

```

Compile e execute o aplicativo. A Figure 4.18 mostra o fundo transparente e o círculo resultante.

Figure 4.18 **BNRHypnosisView** com fundo transparente



## Projeção de círculos concêntricos

Existem duas abordagens que você pode usar para projetar múltiplos círculos concêntricos dentro da **BNRHypnosisView**. Você pode criar múltiplas instâncias de **UIBezierPath**, cada uma representando um círculo. Ou pode adicionar múltiplos círculos à única instância de **UIBezierPath**, e cada círculo será um subcaminho. É ligeiramente mais eficiente usar uma instância; então, será isso que você fará.

Para preencher a tela com círculos concêntricos, você precisa determinar o raio do círculo mais externo. Você começará projetando um círculo com esse raio e, em seguida, projetará círculos com raios decrescentes enquanto o raio permanecer positivo.

Para o raio máximo, você usará metade da hipotenusa da visão inteira. Isso significa que o círculo mais externo quase circunscreverá a visão, e você verá apenas pedaços de cinza-claro nos cantos.

No **BNRHypnosisView.m**, substitua o código que projeta um círculo por um código que projeta círculos concêntricos.

```
- (void)drawRect:(CGRect)rect
{
    CGRect bounds = self.bounds;

    // Figure out the center of the bounds rectangle
    CGPoint center;
    center.x = bounds.origin.x + bounds.size.width / 2.0;
    center.y = bounds.origin.y + bounds.size.height / 2.0;

    // The circle will be the largest that will fit in the view
    float radius = (MIN(bounds.size.width, bounds.size.height) / 2.0);

    // The largest circle will circumscribe the view
    float maxRadius = hypot(bounds.size.width, bounds.size.height) / 2.0;

    UIBezierPath *path = [[UIBezierPath alloc] init];

    // Add an arc to the path at center, with radius of radius,
    // from 0 to 2*PI radians (a circle)
    [path addArcWithCenter:center
                      radius:radius
                 startAngle:0.0
                   endAngle:M_PI * 2.0
                  clockwise:YES];

    for (float currentRadius = maxRadius; currentRadius > 0; currentRadius -= 20) {

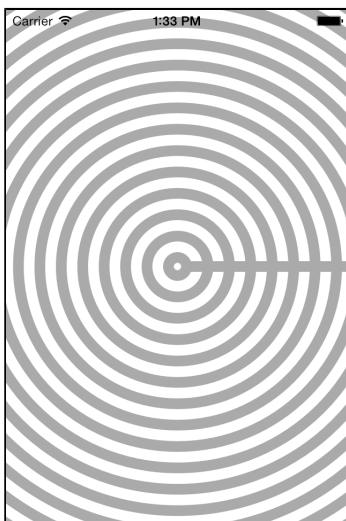
        [path addArcWithCenter:center
                          radius:currentRadius // Note this is currentRadius!
                         startAngle:0.0
                           endAngle:M_PI * 2.0
                          clockwise:YES];
    }

    // Configure line width to 10 points
    path.lineWidth = 10.0;

    // Draw the line!
    [path stroke];
}
```

Compile e execute o aplicativo. Não é exatamente o que você esperava; parecem mais círculos em plantações do que círculos concêntricos (Figure 4.19).

Figure 4.19 **BNRHypnosisView** projetando círculos de plantações



O problema é que seu único objeto **UIBezierPath** está conectando os subcaminhos (os círculos individuais) para formar um caminho completo. Pense em um objeto **UIBezierPath** como um lápis sobre um pedaço de papel –

quando você vai desenhar outro círculo, o lápis continua sobre o pedaço de papel. Você precisa levantar o lápis do papel antes de desenhar um novo círculo.

No loop `for` no `drawRect:` da `BNRHypnosisView`, pegue o lápis e leve-o até o ponto correto antes de desenhar cada círculo.

```
- (void)drawRect:(CGRect)rect
{
    CGRect bounds = self.bounds;

    // Figure out the center of the bounds rectangle
    CGPoint center;
    center.x = bounds.origin.x + bounds.size.width / 2.0;
    center.y = bounds.origin.y + bounds.size.height / 2.0;

    // The largest circle will circumscribe the view
    float maxRadius = hypot(bounds.size.width, bounds.size.height) / 2.0;

    UIBezierPath *path = [[UIBezierPath alloc] init];

    for (float currentRadius = maxRadius; currentRadius > 0; currentRadius -= 20) {

        [path moveToPoint:CGPointMake(center.x + currentRadius, center.y)];

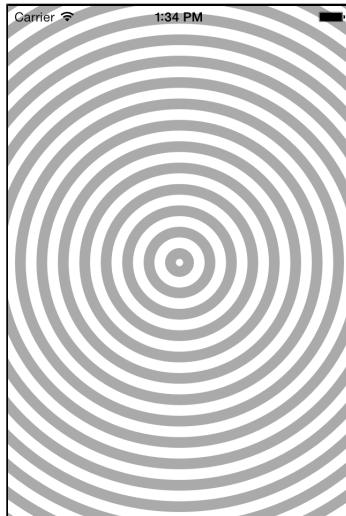
        [path addArcWithCenter:center
                           radius:currentRadius // note this is currentRadius!
                         startAngle:0.0
                           endAngle:M_PI * 2.0
                          clockwise:YES];
    }
}
```

```
// Configure line width to 10 points
path.lineWidth = 10.0;

// Draw the line!
[path stroke];
}
```

Compile e execute o aplicativo. Agora, você já deve ter círculos concêntricos.

Figure 4.20 **BNRHypnosisView** projetando círculos concêntricos



Você viu apenas uma amostra do que **UIBezierPath** pode fazer. Não deixe de examinar a documentação e experimentar alguns dos desafios ao fim deste capítulo para ter uma ideia melhor de algumas das coisas inteligentes que você pode fazer simplesmente encadeando arcos, linhas e curvas.

## Mais documentação do desenvolvedor

A Referência de API, que contém as referências de classe, é uma parte essencial da documentação do desenvolvedor e parte essencial da vida de um desenvolvedor. Mas há mais na documentação do que a Referência de API. A documentação também fornece:

|                   |                                                                                                                                                                   |
|-------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Guias SDK         | organizados por tópico, em vez de por classe ou método, excelentes para aprender mais a respeito de tópicos específicos em desenvolvimento para Objective-C e iOS |
| Código de amostra | projetos pequenos e completos que demonstram como a Apple espera que a classe em questão seja usada                                                               |

Seria difícil exagerar a importância da documentação da Apple para o trabalho diário dos desenvolvedores de iOS. À medida que você lê este livro, reserve um tempo para pesquisar novas classes e métodos conforme os encontra e veja o que mais eles podem fazer. Além disso, leia os guias SDK e faça download de projetos de código de amostra que despertem seu interesse. Você pode ver os guias e códigos de amostra disponíveis na Biblioteca do Desenvolvedor iOS em [developer.apple.com/library](http://developer.apple.com/library).

## Desafio de bronze: desenhar uma imagem

O desafio é carregar uma imagem a partir do sistema de arquivos e projetá-la em cima dos círculos concêntricos, como na Figure 4.21.

Figure 4.21 Projeção de uma imagem



Encontre um arquivo de imagem. Um PNG com algumas partes transparentes seria especialmente interessante. (O arquivo zip que você baixou tem o `logo.png`, que servirá perfeitamente.) Arraste-o para seu projeto Xcode.

Criar um objeto `UIImage` a partir desse arquivo é uma linha:

```
UIImage *logoImage = [UIImage imageNamed:@"logo.png"];
```

Em seu método `drawRect:`, compô-la em sua visão é apenas mais outra:

```
[logoImage drawInRect:someRect];
```

## Para os mais curiosos: Core Graphics

Em geral, o método `drawRect:` usa instâncias de `UIImage`, `UIBezierPath` e `NSString` para projetar imagens, formas e texto, respectivamente. Cada uma dessas classes implementa ao menos um método que, ao ser executado em `drawRect:`, projeta pixels na camada da visão que foi enviada `drawRect:`.

Essas classes fazem o desenho no iOS parecer simples e conveniente. Contudo, há muita coisa acontecendo nos bastidores.

Desenhar imagens no iOS – seja uma imagem que você salvará como JPEG ou PDF, ou uma camada que representa uma `UIView` – é de responsabilidade do framework *Core Graphics*. As classes que você usou para fazer os desenhos neste capítulo, como `UIBezierPath`, englobam código do Core Graphics em seus métodos para facilitar o desenho para o programador. Para entender de fato como essas classes funcionam e como as imagens são criadas, você deve entender como o Core Graphics faz seu trabalho.

Core Graphics é um API de desenho 2D construído em C. Assim, não há nenhum objeto ou método Objective-C, mas, em vez disso, estruturas em C e funções em C que imitam o comportamento orientado ao objeto. O “objeto” Core Graphics mais importante é o *contexto de gráficos*, que, na realidade, contém duas coisas: o estado do desenho, tal como a cor atual da caneta e a espessura da linha, e a memória sobre a qual ele está sendo projetado. Um contexto de gráficos é representado por “instâncias” de `CGContextRef`.

Logo antes do `drawRect:` ser enviado a uma instância de `UIView`, o sistema cria um `CGContextRef` para a camada daquela visão. A camada tem os mesmos bounds que a visão e alguns valores padrão para seu estado de desenho. À medida que as operações de desenho são enviadas para o contexto, os pixels da camada são alterados. Depois de o `drawRect:` ser concluído, o sistema pega a camada e a compõe na tela.

Todas as classes de desenho que você usou neste capítulo sabem como chamar funções de Core Graphics que alteram o estado do desenho e emitem operações de desenho sobre o `CGContextRef` adequado. Por exemplo, enviar `setStroke` a uma instância de `UIColor` chamará funções que alteram o estado do desenho do contexto atual. Sendo assim, esses dois pedaços de código são equivalentes:

```
[[UIColor colorWithRed:1.0 green:0.0 blue:1.0 alpha:1.0] setStroke];
UIBezierPath *path = [UIBezierPath bezierPath];
[path moveToPoint:a];
[path addLineToPoint:b];
[path stroke];
```

É equivalente a estas linhas:

```
CGContextSetRGBStrokeColor(currentContext, 1, 0, 0, 1);
CGMutablePathRef path = CGPathCreateMutable();
CGPathMoveToPoint(path, NULL, a.x, a.y);
CGPathAddLineToPoint(path, NULL, b.x, b.y);
CGContextAddPath(currentContext, path);

CGContextStrokePath(currentContext);
CGPathRelease(path);
```

As funções Core Graphics que operam no contexto, tal como **CGContextSetRGBStrokeColor**, levam um ponteiro para o contexto que elas modificarão como seu primeiro argumento. Você pode pegar um ponteiro para o contexto atual no **drawRect**: ao chamar a função **UIGraphicsGetCurrentContext**. O contexto atual é um ponteiro que engloba todo o aplicativo e que está definido para apontar para o contexto criado para uma visão logo antes de a visão receber o **drawRect**:

```
- (void)drawRect:(CGRect)rect
{
    CGContextRef currentContext = UIGraphicsGetCurrentContext();

    CGContextSetRGBStrokeColor(currentContext, 1, 0, 0, 1);

    CGMutablePathRef path = CGPathCreateMutable();
    CGPathMoveToPoint(path, NULL, a.x, a.y);
    CGPathAddLineToPoint(path, NULL, b.x, b.y);
    CGContextAddPath(currentContext, path);

    CGContextStrokePath(currentContext);
    CGPathRelease(path);

    CGContextSetStrokeColorWithColor(currentContext, color);
}
```

Qualquer coisa que você possa fazer com **UIBezierPath** e **UIColor** pode ser feita diretamente no Core Graphics. Na verdade, há estruturas em C que têm o mesmo comportamento dessas classes (**CGMutablePathRef** e **CGColorRef**). Contudo, costuma ser mais fácil trabalhar com contrapartes Objective-C.

Além disso, existem algumas coisas que você simplesmente ainda não pode fazer sem descer ao Core Graphics, como projetar gradientes. No entanto, como você se lembrou que todos os tipos de frameworks têm o mesmo prefixo, você pode buscar na documentação os tipos que começam com CG para descobrir o que está disponível para você.

Você pode estar se perguntando por que muitos dos tipos de Core Graphics têm um Ref depois deles. Todo tipo de Core Graphics é uma estrutura, mas alguns imitam o comportamento de objetos ao serem alocados para o heap. Portanto, quando você cria um desses “objetos” Core Graphics, recebe um ponteiro para o endereço deles na memória.

Cada estrutura Core Graphics que é alocada dessa maneira tem uma definição de tipo que incorpora o asterisco (\*) ao próprio tipo. Por exemplo, existe uma estrutura **CGColor** (que você nunca usa) e uma definição de tipo **CGColorRef** que significa **CGColor \*** (que você sempre usa). Essa convenção facilita para o programador olhar rapidamente para o código e determinar se a variável é ou não uma estrutura em C disfarçada de objeto ou um objeto em Objective-C ao qual você pode enviar mensagens.

Outro ponto de confusão para programadores no Core Graphics é o fato de alguns tipos não terem uma Ref ou um asterisco, como **CGRect** e **CGPoint**. Esses tipos são pequenas estruturas de dados que podem conviver na pilha, e, por isso, não há necessidade de passar um ponteiro para eles.

Entretanto, alguns tipos de Core Graphics são muito mais envolvidos do que simplesmente o fato de se fixarem a alguns *floats* – eles, na verdade, têm ponteiros para outros objetos Core Graphics. Esses “objetos” tomarão forte posse dos objetos para os quais apontam, mas ARC não rastreará essa posse. Em vez disso, você precisa liberar manualmente a posse desses tipos de objetos quando tiver terminado com eles. A regra é: se você criar um objeto Core Graphics com uma função que tenha a palavra *Create* ou *Copy* nele, você deve chamar a função *Release* que combina com ele e passar um ponteiro para o objeto como o primeiro argumento.

Um lembrete final a respeito do Core Graphics: Ele também existe no Mac. Você pode construir códigos, como se faz no framework core-plot de código aberto, que funcionarão tanto no iOS quanto no OS X.

## Desafio de ouro: sombras e gradientes

Neste momento, adicionar projeção de sombra e desenhar com gradientes só pode ser feito usando Core Graphics.

Para criar uma projeção de sombra, você instala uma sombra no contexto de gráficos. Depois disso, qualquer coisa opaca que você desenhar terá uma projeção de sombra. A sombra tem um deslocamento (que é expresso com um *CGSize*) e um borrão em pontos. Esta é a declaração do método usado para instalar a sombra no contexto de gráficos:

```
void CGContextSetShadow (CGContextRef context, CGSize offset, CGFloat blur);
```

(Há uma versão que recebe uma cor, mas você quase sempre quer uma sombra escura.)

Não há função para remover sombras. Assim, você precisará salvar o estado dos gráficos antes de configurar a sombra e, depois, restaurá-lo após configurar a sombra. Isso será semelhante ao seguinte:

```
CGContextSaveGState(currentContext);
CGContextSetShadow(currentContext, CGSizeMake(4,7), 3);

// Draw stuff here, it will appear with a shadow

CGContextRestoreGState(currentContext);

// Draw stuff here, it will appear with no shadow
```

A primeira parte do desafio é colocar uma projeção de sombra na imagem que você compôs sobre a visão no desafio anterior.

Figure 4.22 Projeção de sombra



Gradientes permitem a você fazer um sombreado que percorre facilmente uma lista de cores. O *CGGradientRef* tem uma lista de cores, e você solicita para que ele desenhe a lista de maneira linear ou radial. Isso será semelhante ao seguinte:

```
CGFloat locations[2] = { 0.0, 1.0 };
CGFloat components[8] = { 1.0, 0.0, 0.0, 1.0,    // Start color is red
                        1.0, 1.0, 0.0, 1.0 }; // End color is yellow

CGColorSpaceRef colorspace = CGColorSpaceCreateDeviceRGB();
CGBitmapInfo bitmapInfo = kCGImageAlphaNoneSkipFirst;
CGColorSpaceRelease(colorspace);

CGBitmapContextRef context = CGBitmapContextCreate(NULL,
                                                    width, height,
                                                    bitsPerComponent, bytesPerRow,
                                                    colorspace, bitmapInfo);
CGContextRef currentContext = CCGraphicsGetCurrentContext();
CGContextSetAllowsAntialiasing(currentContext, YES);
CGContextSetShouldAntialias(currentContext, YES);

CGContextSetFillColorWithColor(context, redColor);
CGContextFillRect(context, rect);
CGContextRelease(context);

CGContextDrawLinearGradient(currentContext, gradient, startPoint, endPoint, 0);
CGGradientRelease(gradient);
CGColorSpaceRelease(colorspace);
```

O último argumento para `CGContextDrawLinearGradient()` determina o que acontece antes do ponto inicial e após o ponto final. Se você quiser que a primeira cor cubra o espaço antes do ponto inicial, você fornece `kCGGradientDrawsBeforeStartLocation`. Se você quiser que a última cor cubra o espaço após o ponto final, você fornece `kCGGradientDrawsAfterEndLocation`. Para usar ambos, de forma binária ou juntas:

```
CGContextDrawLinearGradient(currentContext, gradient, startPoint, endPoint,
                           kCGGradientDrawsBeforeStartLocation | kCGGradientDrawsAfterEndLocation);
```

O complicado em relação a gradientes é o fato de eles cobrirem tudo na visão. Antes de projetar um gradiente, você costuma instalar um caminho de corte no contexto de gráficos que define o que você quer que seja pintado no gradiente. Depois, você projeta o gradiente. Novamente, não há função para limpar o caminho de corte; sendo assim, você costuma salvar o estado dos gráficos antes de instalar o caminho de corte e restaura o estado depois.

Se tiver um `CGContextRef` e um `UIBezierPath`, é assim que você instala esse caminho como caminho de corte:

```
CGContextSaveGState(currentContext);
[myPath addClip];

// Draw your gradient here

CGContextRestoreGState(currentContext);
```

O desafio é preencher um triângulo com um gradiente que vá do amarelo na parte de baixo ao verde na parte de cima.

Figure 4.23 Triângulo de gradiente



# 5

## Visões: redesenho e UIScrollView

Neste capítulo, você verá como visões são redesenhasadas em resposta a um evento. Em especial, você atualizará o Hypnosister para que, quando o usuário tocar na **BNRHypnosisView**, a cor de seu círculo mude. Uma mudança na cor que exigirá que a própria visão se redesenhe. Posteriormente neste capítulo, você também acrescentará uma **UIScrollView** à hierarquia de visão do Hypnosister.

O primeiro passo é declarar uma propriedade para a cor em **BNRHypnosisView**. Em aplicativos anteriores, você declarava propriedades em arquivos de cabeçalho. Você pode também declarar propriedades nas *extensões de classe*.

Abra o `BNRHypnosis.m` e acrescente o seguinte código próximo do topo do arquivo.

```
#import "BNRHypnosisView.h"

@interface BNRHypnosisView : UIView

@property (strong, nonatomic) UIColor *circleColor;

@end

@implementation BNRAppDelegate
```

Estas três linhas de código são uma extensão de classe com uma declaração de propriedade. Por que essa propriedade é declarada em uma extensão de classe e não no arquivo de cabeçalho? Guarde essa pergunta, e nós já voltaremos a ela depois de você terminar de implementar a mudança de cor. Enquanto isso, pense em `circleColor` como apenas outra propriedade de **BNRHypnosisView**.

No `BNRHypnosis.m`, atualize o método **initWithFrame:** para criar uma `circleColor` padrão para instâncias de **BNRHypnosisView**.

```
- (instancetype)initWithFrame:(CGRect)frame
{
    self = [super initWithFrame:frame];
    if (self) {
        self.backgroundColor = [UIColor clearColor];
        self.circleColor = [UIColor lightGrayColor];
    }
    return self;
}
```

No **drawRect:**, modifique a mensagem que estabelece a cor atual de pincelada a ser usada `circleColor`.

```
// Configure line width to 10 points
path.lineWidth = 10;

[[UIColor lightGrayColor] setStroke];
[self.circleColor setStroke];

// Draw the line!
[path stroke];
```

Você pode compilar e executar o aplicativo para confirmar se funciona como antes. O próximo passo é escrever o código que atualizará `circleColor` quando a visão for tocada.

Quando o usuário toca em uma visão, a mensagem **touchesBegan:withEvent:** é enviada à visão. O método **touchesBegan:withEvent:** é um handler de evento de toque. Você verá eventos de toque e handlers

de eventos de toque em mais detalhes no Chapter 12. No momento, você simplesmente sobrescreverá **touchesBegan:withEvent:** para alterar a propriedade `circleColor` da visão para uma cor aleatória.

No `BNRHypnosisView.m`, sobrescreva **touchesBegan:withEvent:**: para gerar uma mensagem de registro, crie uma `UIColor` de cor aleatória e defina `circleColor` para essa cor.

```
// When a finger touches the screen
- (void)touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event
{
    NSLog(@"%@", self);

    // Get 3 random numbers between 0 and 1
    float red = (arc4random() % 100) / 100.0;
    float green = (arc4random() % 100) / 100.0;
    float blue = (arc4random() % 100) / 100.0;

    UIColor *randomColor = [UIColor colorWithRed:red
                                              green:green
                                                blue:blue
                                               alpha:1.0];

    self.circleColor = randomColor;
}
```

Compile e execute o aplicativo e toque em qualquer lugar da visão. Sua mensagem aparecerá no console, mas a cor do círculo não mudará. Sua visão não está sendo redesenhada. Para entender por que e como resolver o problema, você precisa saber a respeito do loop de execução.

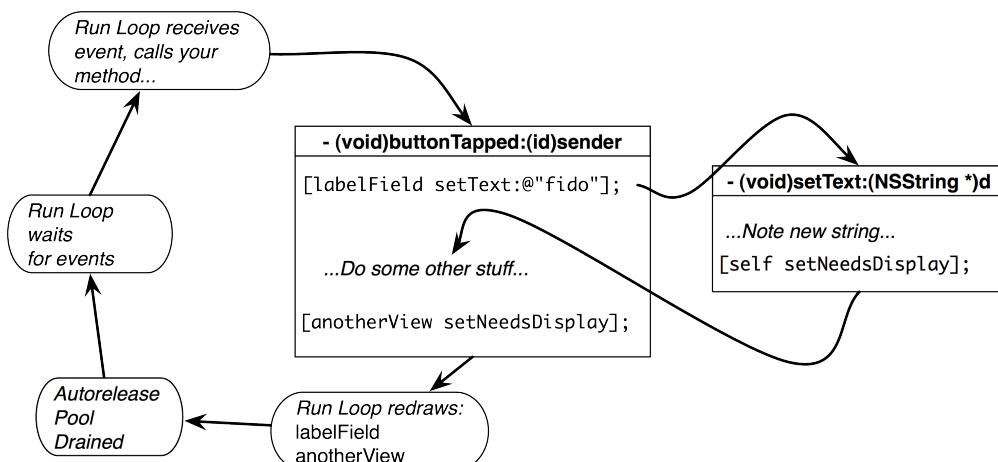
## Loop de execução e redesenho de visões

Quando um aplicativo do iOS é iniciado, ele dá início a um *loop de execução*. O trabalho do loop de execução é prestar atenção nos eventos, como um toque. Quando um evento ocorre, o loop de execução encontra os métodos handler adequados para o evento. Esses métodos handler chamam outros métodos, que chamam mais métodos, e assim por diante. Após todos os métodos serem concluídos, o controle volta ao loop de execução.

Quando o loop de execução recupera o controle, ele verifica uma lista de “visões sujas” – visões que precisam ser novamente renderizadas com base no que aconteceu na rodada mais recente de manejo de eventos. O loop de execução envia, então, a mensagem **drawRect:** para as visões desta lista antes de todas as visões da hierarquia serem compostas juntas novamente.

A Figure 5.1 mostra onde acontece o redesenho da tela no loop de execução usando um exemplo do usuário digitando texto em um campo de texto.

Figure 5.1 Redesenho de visões com o loop de execução



Estas duas otimizações – apenas renderizar novamente visões que precisem disso e enviar **drawRect:** apenas uma vez por evento – mantêm as interfaces do iOS ágeis. Se os aplicativos do iOS precisarem redesenhar todas as visões toda vez que um evento foi processado, haveria um desperdício muito grande de tempo fazendo

um trabalho desnecessário. Agrupar o redesenho de visões ao fim de um ciclo de loop de execução evita o redesenho desnecessário de uma visão mais de uma vez se mais de uma de suas propriedades forem alteradas em um único evento.

Vamos dar uma olhada no que está acontecendo no Hypnosister. Você sabe que seu evento de toque está sendo corretamente encaminhado ao handler de toque da **BNRHypnosisView** porque você vê sua mensagem de registro. Mas, quando **touchesBegan:withEvent:** termina de executar e o controle retorna ao loop de execução, o loop de execução não envia **drawRect:** à **BNRHypnosisView**.

Para obter uma visão da lista de visões sujas, você deve enviar a ela a mensagem **setNeedsDisplay**. As subclasses da **UIView** que fazem parte do SDK do iOS enviam **setNeedsDisplay** a si próprias sempre que seu conteúdo muda. Por exemplo, uma instância de **UILabel** enviará **setNeedsDisplay** a si própria quando receber **setText:**, já que mudar o texto de um rótulo exige que o rótulo renderize novamente sua camada. Em subclasses de **UIView** personalizadas, como **BNRHypnosisView**, você mesmo deve enviar essa mensagem.

No **BNRHypnosisView.m**, implemente um acessor personalizado para a propriedade **circleColor** para enviar **setNeedsDisplay** à visão sempre que essa propriedade for alterada.

```
- (void)setCircleColor:(UIColor *)circleColor
{
    _circleColor = circleColor;
    [self setNeedsDisplay];
}
```

Compile e execute o aplicativo novamente. Toque na visão, e a cor do círculo mudará.

(Há outra otimização possível ao redesenhar: você pode marcar apenas uma parte da visão como sendo de redesenho necessário. Isso é feito ao se enviar **setNeedsDisplayInRect:** a uma visão. Quando **drawRect:** é enviado para a visão suja, o argumento para esse método que vimos esse tempo todo ignorando será o retângulo passado para **setNeedsDisplayInRect:**. No geral, você não ganha tanto desempenho assim e acaba fazendo um trabalho difícil para conseguir fazer esse comportamento de redesenho parcial funcionar direito; por isso, a maioria das pessoas não se dá ao trabalho, a menos que seu código de desenho esteja claramente deixando o aplicativo lento.)

## Extensões de classe

Agora, vamos voltar à propriedade **circleColor** que você declarou em uma extensão de classe para **BNRHypnosisView**. Qual é a diferença entre uma propriedade declarada em uma extensão de classe e uma declarada em um arquivo de cabeçalho? Visibilidade.

Um arquivo de cabeçalho de classe é visível a outras classes. Essa, na verdade, é a finalidade dele. Uma classe declara propriedades e métodos em seu arquivo de cabeçalho para anunciar a outras classes como elas podem interagir com a classe ou suas instâncias.

No entanto, nem toda propriedade ou método é para consumo público. Propriedades e métodos que sejam usados internamente pela classe pertencem a uma extensão de classe. A propriedade **circleColor** é usada apenas pela classe **BNRHypnosisView**. Nenhuma outra classe precisa saber dessa propriedade. Assim, ela pertence a uma extensão de classe.

Colocar propriedades e métodos em uma extensão de classe não é ser paranoico ou exageradamente possessivo. É uma boa prática manter seu arquivo de cabeçalho o mais sucinto possível. Isso facilita para que outros compreendam como podem usar sua classe.

Sintaticamente, uma extensão de classe se parece um pouco com um arquivo de cabeçalho. Ela começa com **@interface** seguido por um conjunto de parênteses vazios. O **@end** marca o final da extensão de classe.

Tipicamente, você coloca a extensão de classe no topo do arquivo de implementação antes de a palavra-chave **@implementation** anunciar o início das definições do método.

```
#import "BNRHypnosisView.h"

@interface BNRHypnosisView ()

@property (strong, nonatomic) UIColor *circleColor;

@end

@implementation BNRAppDelegate
```

As mesmas regras de visibilidade se mantêm para as subclasses. Se você quisesse subclassificar **BNRHypnosisView**, a subclasse e suas instâncias não teriam conhecimento de `circleColor`.

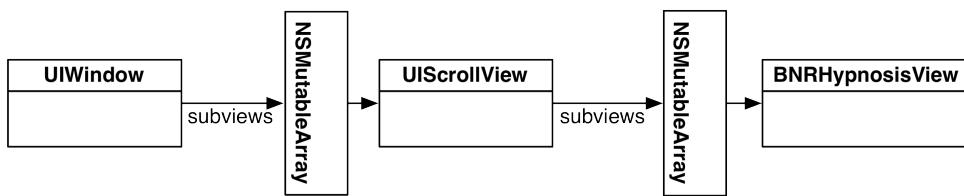
Se você precisar de visibilidade limitada para determinadas propriedades e métodos, pode criar uma extensão de classe em um arquivo externo e importá-lo para os arquivos de implementação de classes de acordo com o que for necessário saber.

Usaremos extensões de classe da maneira adequada durante todo o livro para ocultar detalhes de implementação que não precisam ficar visíveis fora da classe.

## Utilização de UIScrollView

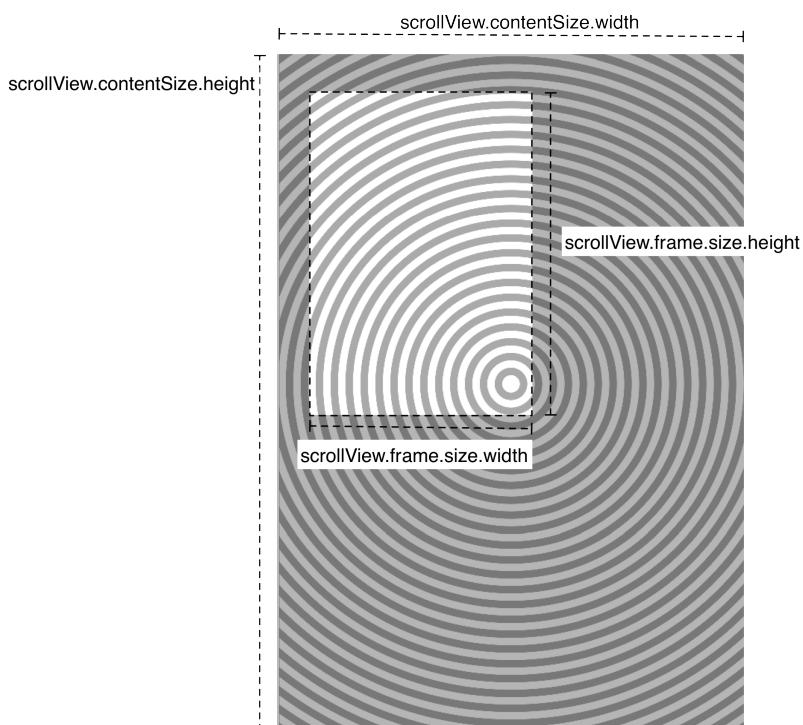
Nesta seção, você vai adicionar uma instância de **UIScrollView** ao Hypnosister. Essa visão com rolagem será uma subvisão direta da janela, e a instância de **BNRHypnosisView** será uma subvisão da visão com rolagem, conforme mostrado na Figure 5.2.

Figure 5.2 Hierarquia de visão com **UIScrollView**



Visões com rolagem são tipicamente usadas para visões que são maiores que a tela. Uma visão com rolagem desenha uma parte retangular de sua subvisão e, quando você move seu dedo ou arrasta a tela, isso faz mudar a posição desse retângulo na subvisão. Portanto, você pode pensar na visão com rolagem como uma porta de visualização que você pode movimentar (Figure 5.3). O tamanho da visão com rolagem é o tamanho dessa porta de visualização. O tamanho da área que ela pode ser usada para visualizar é o `contentSize` da **UIScrollView**, que é tipicamente o tamanho da subvisão da **UIScrollView**.

Figure 5.3 **UIScrollView** e sua área de conteúdo



**UIScrollView** é uma subclasse da **UIView**, portanto, pode ser inicializada usando **initWithFrame:** e pode ser adicionada como uma subvisão a outra visão.

No **BNRAppDelegate.m**, coloque uma versão superdimensionada da **BNRHypnosisView** dentro de uma visão com rolagem e adicione essa visão com rolagem à janela:

```
- (BOOL)application:(UIApplication *)application
didFinishLaunchingWithOptions:(NSDictionary *)launchOptions
{
    self.window = [[UIWindow alloc] initWithFrame:[[UIScreen mainScreen] bounds]];
    // Override point for customization after application launch

    CGRect firstFrame = self.window.bounds;
    BNRHypnosisView *firstView = [[BNRHypnosisView alloc] initWithFrame:firstFrame];
    [self.window addSubview:firstView];

    // Create CGRects for frames
    CGRect screenRect = self.window.bounds;
    CGRect bigRect = screenRect;
    bigRect.size.width *= 2.0;
    bigRect.size.height *= 2.0;

    // Create a screen-sized scroll view and add it to the window
    UIScrollView *scrollView = [[UIScrollView alloc] initWithFrame:screenRect];
    [self.window addSubview:scrollView];

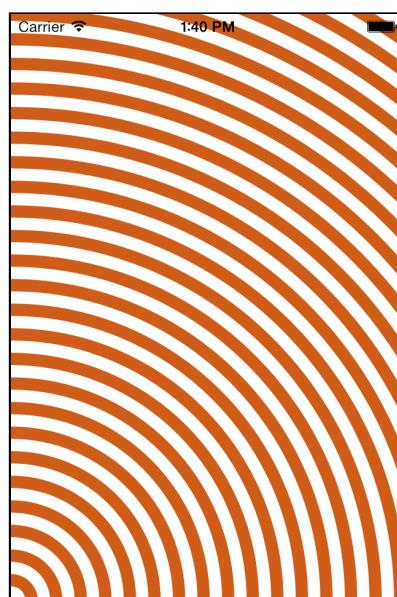
    // Create a super-sized hypnosis view and add it to the scroll view
    BNRHypnosisView *hypnosisView = [[BNRHypnosisView alloc] initWithFrame:bigRect];
    [scrollView addSubview:hypnosisView];

    // Tell the scroll view how big its content area is
    scrollView.contentSize = bigRect.size;

    self.window.backgroundColor = [UIColor whiteColor];
}
```

Compile e execute seu aplicativo. Você pode arrastar sua visão para cima e para baixo, para a esquerda e para a direita para ver mais da **BNRHypnosisView** superdimensionada.

Figure 5.4 Quadrante superior direito da **BNRHypnosisView** grande



Quando você vai arrastar a **BNRHypnosisView**, a cor do círculo muda. Você não pode arrastar sem dar início a um toque; então, o loop de execução envia eventos de toque para a **UIScrollView** e para a **BNRHypnosisView**. No Chapter 13, você verá como reconhecer e lidar com um gesto de “toque rápido”, para que ele possa ser diferenciado de um toque ou arrasto.

“Pinçar para zoom” também é implementado usando **UIScrollView**. Não são necessárias muitas linhas de código, mas isso envolve uma técnica que ainda não abordamos. Então, acrescentar pinçar para zoom ao Hypnosister será um desafio na Chapter 7.

## Arraste de tela e paginação

Outro uso para uma visão com rolagem é arrastar a tela entre diversas instâncias de visão.

No `BNRAppDelegate.m`, reduza a **BNRHypnosisView** de volta ao tamanho da tela e adicione uma segunda **BNRHypnosisView** do tamanho da tela como outra subvisão da **UIScrollView**. Configure o `contentSize` da visão com rolagem para que tenha duas vezes a largura da tela, mas com a mesma altura.

```
// Create CGRects for frames
CGRect screenRect = self.window.bounds;
CGRect bigRect = screenRect;
bigRect.size.width *= 2.0;
bigRect.size.height *= 2.0;

// Create a screen-sized scroll view and add it to the window
UIScrollView *scrollView = [[UIScrollView alloc] initWithFrame:screenRect];
[self.window addSubview:scrollView];

// Create a super-sized hypnosis view and add it to the scroll view
BNRHypnosisView *hypnosisView = [[BNRHypnosisView alloc] initWithFrame:bigRect];
// Create a screen-sized hypnosis view and add it to the scroll view
BNRHypnosisView *hypnosisView = [[BNRHypnosisView alloc] initWithFrame:screenRect];
[scrollView addSubview:hypnosisView];

// Add a second screen-sized hypnosis view just off screen to the right
screenRect.origin.x += screenRect.size.width;
BNRHypnosisView *anotherView = [[BNRHypnosisView alloc] initWithFrame:screenRect];
[scrollView addSubview:anotherView];

// Tell the scroll view how big its content area is
scrollView.contentSize = bigRect.size;
```

Compile e execute o aplicativo. Arraste a tela da esquerda para a direita para ver cada instância de **BNRHypnosisView**. Observe que você pode parar entre as duas visões.

Figure 5.5 Entre as duas visões de hipnose



Às vezes, você quer isso, mas, às vezes, não quer. Para forçar a visão com rolagem a fixar sua porta de visualização em uma das visões, ative a paginação para a visão com rolagem no `BNRAppDelegate.m`.

```
UIScrollView *scrollView = [[UIScrollView alloc] initWithFrame:screenRect];
scrollView.pagingEnabled = YES;
[self.window addSubview:scrollView];
```

Compile e execute o aplicativo. Arraste a tela para o meio das duas visões e observe como ela se encaixa em uma visão ou na outra. A paginação funciona pegando o tamanho de bounds da visão com rolagem e dividindo o contentSize que ela exibe em seções do mesmo tamanho. Após o usuário arrastar a tela, a porta de visualização rolará para mostrar apenas uma dessas seções.



# 6

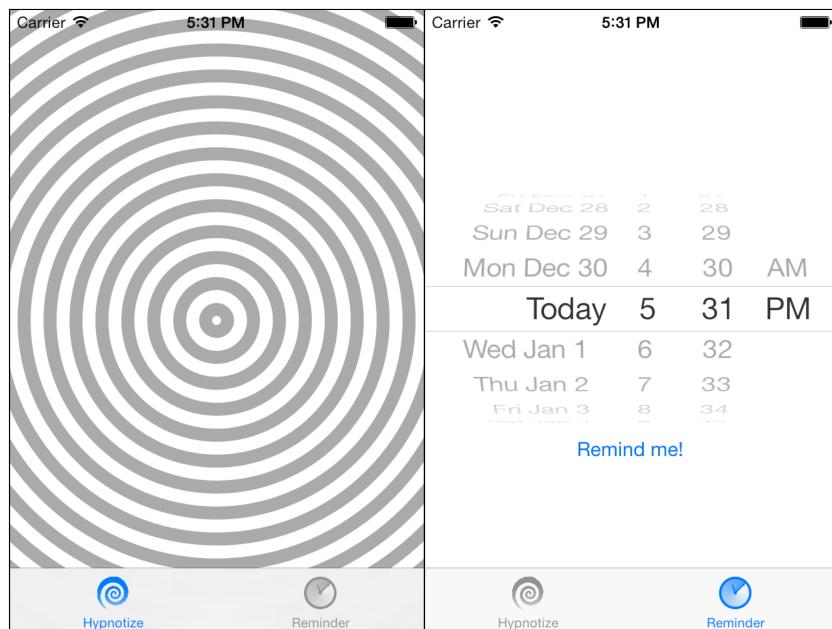
## Controladores de visão

No Chapter 5, você criou uma hierarquia de visão (uma visão com rolagem com duas subvisões) e a apresentou na tela adicionando explicitamente a visão com rolagem como uma subvisão da janela do aplicativo. É mais comum fazer isso usando um *controlador de visão*.

Um controlador de visão é uma instância de uma subclasse de **UIViewController**. Um controlador de visão gerencia uma hierarquia de visão. Ele é responsável por criar objetos de visão que compõem a hierarquia, por lidar com eventos associados aos objetos de visão em sua hierarquia e por adicionar sua hierarquia à janela.

Neste capítulo, você criará um aplicativo chamado HypnoNerd. No HypnoNerd, o usuário será capaz de alternar entre duas hierarquias de visão – uma para ser hipnotizado e a outra para configurar um lembrete de hipnose para uma data futura.

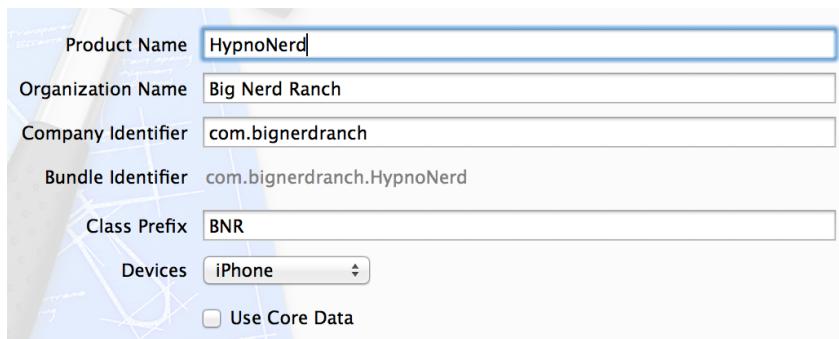
Figure 6.1 As duas faces do HypnoNerd



Para fazer isso acontecer, você criará duas subclasses **UIViewController**: **BNRHypnosisViewController** e **BNRReminderViewController**. Você usará a classe **UITabBarController** para permitir ao usuário alternar entre as hierarquias de visão dos dois controladores de visão.

Crie um novo projeto iOS (Command-Shift-N) a partir do template Empty Application. Nomeie esse projeto como HypnoNerd e configure-o conforme mostrado na Figure 6.2.

Figure 6.2 Criação de um novo projeto

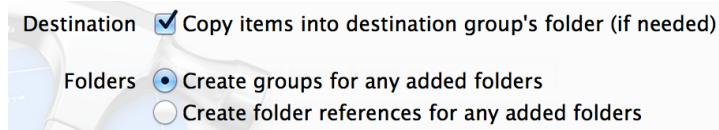


Você vai reutilizar a classe **BNRHypnosisView** do Hypnosister neste projeto.

No Finder, localize o diretório que contém seu projeto do Hypnosister. Arraste os arquivos **BNRHypnosisView.h** e **BNRHypnosisView.m** do Finder para o navegador do projetos no Xcode.

Na folha que aparece, marque a caixa de **Copy items into destination group's folder (if needed)** e caixa ao lado do destino **HypnoNerd** e clique em **Finish** (Figure 6.3).

Figure 6.3 Cópia de arquivos para o HypnoNerd

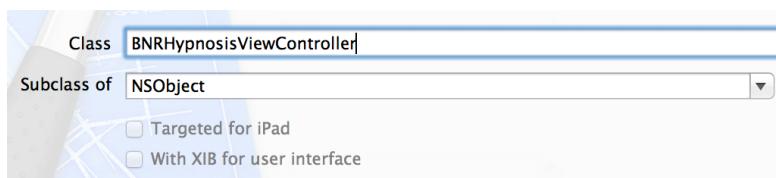


Isso criará cópias dos dois arquivos e os acrescentará ao diretório do HypnoNerd no sistema de arquivos e ao projeto HypnoNerd.

## Criação de subclasses UIViewController

No menu **File**, selecione **New → File...**. Na seção **iOS**, selecione **Cocoa Touch** e, em seguida, escolha **Objective-C class**. Clique em **Next**.

Nomeie essa classe **BNRHypnosisViewController** e escolha **NSObject** como sua superclasse (Figure 6.4). Clique em **Next** e salve os arquivos para concluir a criação da classe.

Figure 6.4 Criação de **BNRHypnosisViewController**

Você criou a classe com o template **NSObject** para começar com o template mais simples possível. Ao começar de forma simples, você tem a oportunidade de ver como as peças funcionam juntas.

Abra o **BNRHypnosisViewController.h** e altere a superclasse para **UIViewController**.

```
@interface BNRHypnosisViewController : NSObject
@interface BNRHypnosisViewController : UIViewController
@end
```

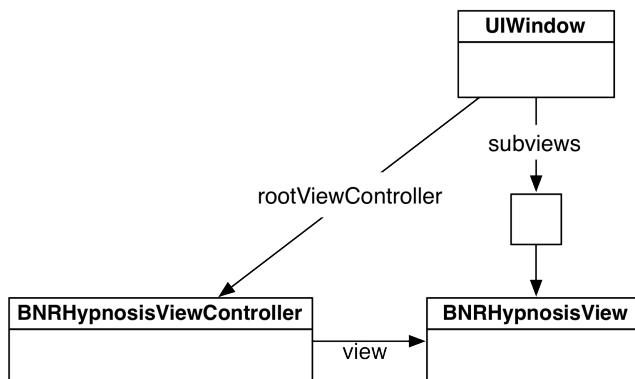
## A visão de um controlador de visão

Como uma subclasse de **UIViewController**, **BNRHypnosisViewController** herda uma importante propriedade:

```
@property (nonatomic, strong) UIView *view;
```

Essa propriedade aponta para uma instância **UIView** que é a raiz da hierarquia de visão do controlador de visão. Quando a view de um controlador de visão é adicionada como uma subvisão da janela, toda a hierarquia de visão do controlador de visão é adicionada.

Figure 6.5 Diagrama de objetos do HypnoNerd



A view de um controlador de visão não é criada até que ela precise aparecer na tela. Essa otimização é chamada de *carregamento lento* e, com frequência, pode conservar memória e melhorar o desempenho.

Há duas maneiras como um controlador de visão pode criar sua hierarquia de visão:

- programaticamente, sobrescrevendo o método **UIViewController loadView**.
- no Interface Builder, carregando um arquivo NIB. (Lembre-se de que um arquivo NIB é o arquivo que é carregado e o arquivo XIB é o que você edita no Interface Builder.)

Como a hierarquia de visão de **BNRHypnosisViewController** consiste em apenas uma visão, ela é uma boa candidata para ser criada programaticamente.

## Criação de uma visão programaticamente

Abra **BNRHypnosisViewController.m** e importe o arquivo de cabeçalho para **BNRHypnosisView**. Em seguida, sobrescreva **loadView** para criar uma instância do tamanho da tela de **BNRHypnosisView** e configure-a como a view do controlador de visão.

```
#import "BNRHypnosisViewController.h"
#import "BNRHypnosisView.h"

@implementation BNRHypnosisViewController

- (void)loadView
{
    // Create a view
    BNRHypnosisView *backgroundView = [[BNRHypnosisView alloc] init];

    // Set it as *the* view of this view controller
    self.view = backgroundView;
}

@end
```

Quando um controlador de visão é criado, sua propriedade **view** é **nil**. Se perguntarem a um controlador de visão sua **view** e a **view for nil**, então uma mensagem **loadView** é enviada ao controlador de visão.

O passo seguinte é adicionar a hierarquia de visão de **BNRHypnosisViewController** à janela do aplicativo de maneira que ela apareça na tela para os usuários.

## Configuração do controlador de visão raiz

Há um método conveniente de adicionar a hierarquia de visão de um controlador de visão à janela: O **setRootViewController:** da **UIWindow**. Configurar um controlador de visão como **rootViewController** adiciona a **view** desse controlador de visão como uma subvisão da janela. Isso automaticamente também redimensiona a **view** para ser do mesmo tamanho da janela.

No **BNRAppDelegate.m**, importe **BNRHypnosisViewController.h** no topo do arquivo. Em seguida, crie uma instância de **BNRHypnosisViewController** e defina-a como o **rootViewController** da janela.

```
#import "BNRAppDelegate.h"
#import "BNRHypnosisViewController.h"

@implementation BNRAppDelegate

- (BOOL)application:(UIApplication *)application
    didFinishLaunchingWithOptions:(NSDictionary *)launchOptions
{
    self.window = [[UIWindow alloc] initWithFrame:[[UIScreen mainScreen] bounds]];
    // Override point for customization after application launch

    BNRHypnosisViewController *hvc = [[BNRHypnosisViewController alloc] init];
    self.window.rootViewController = hvc;

    self.window.backgroundColor = [UIColor whiteColor];
    [self.window makeKeyAndVisible];
    return YES;
}
```

A **view** do controlador de visão raiz aparece no início da execução do aplicativo. Assim, a janela a solicita quando configura o controlador de visão como seu **rootViewController**.

Com base no que você aprendeu no Chapter 4, você pode imaginar a aparência do núcleo do **setRootViewController:**:

```
- (void)setRootViewController:(UIViewController *)viewController
{
    // Get the view of the root view controller
    UIView *rootView = viewController.view;

    // Make a frame that fits the window's bounds
    CGRect viewFrame = self.bounds;
    rootView.frame = viewFrame;

    // Insert this view as window's subview
    [self addSubview:rootView];

    // Update the instance variable
    _rootViewController = viewController;
}
```

No início dessa implementação, foi solicitado a **BNRHypnosisViewController** sua **view**. Como a **BNRHypnosisViewController** acabou de ser criada, sua **view** é **nil**. Sendo assim, é enviada a ela a mensagem **loadView** que cria sua **view**.

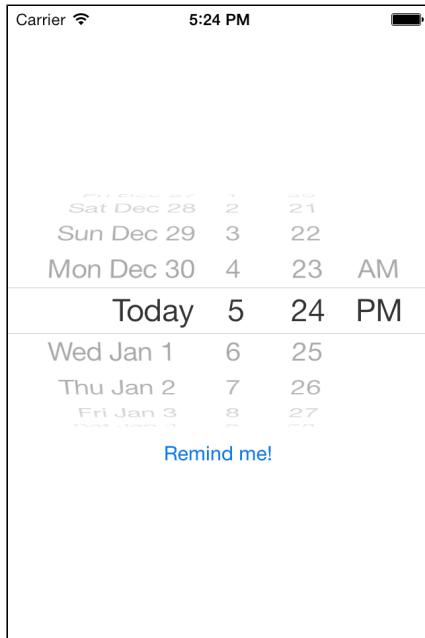
Compile e execute o aplicativo. O HypnoNerd tem a aparência muito semelhante ao Hypnosister. Nos bastidores, porém, é bastante diferente. Você está usando um controlador de visão para apresentar a **BNRHypnosisView** em vez de adicionar o objeto de visão em si à janela. Isso adiciona uma camada de complexidade, que, como você verá mais para o final do capítulo, dá a você poder e flexibilidade para fazer coisas legais.

## Outra UIViewController

Nesta seção, você criará a classe **BNRReminderViewController**. Eventualmente, esse controlador de visão permitirá ao usuário escolher uma data na qual receber um lembrete para ser hipnotizado. Esse lembrete

assumirá a forma de uma notificação que aparecerá mesmo se o HypnoNerd não estiver sendo executado no momento.

Figure 6.6 **BNRReminderViewController**



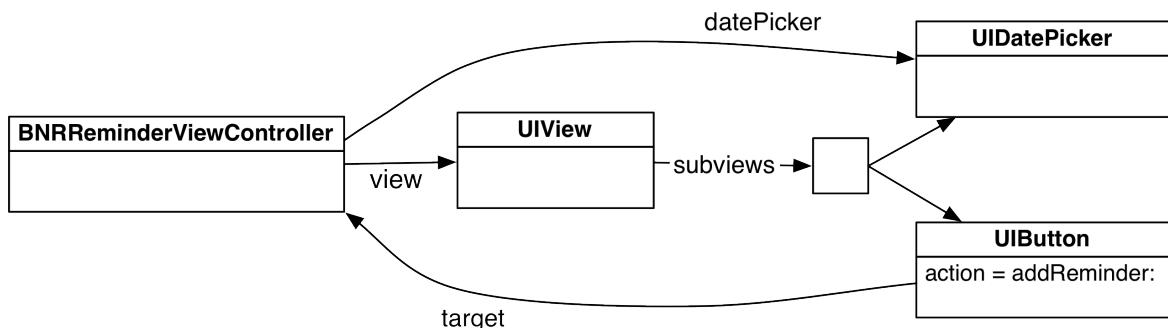
Crie uma nova classe do Objective-C (Command-N). Chame-a de **BNRReminderViewController** e torne-a uma subclasse de **NSObject**.

No **BNRReminderViewController.h**, altere a superclasse para **UIViewController**.

```
@interface BNRReminderViewController : NSObject
@interface BNRReminderViewController : UIViewController
```

A view de **BNRReminderViewController** será uma **UIView** de tela cheia com duas subvisões – uma instância de **UIDatePicker** e uma instância de **UIButton** (Figure 6.7).

Figure 6.7 Diagrama de objetos da hierarquia de visão de **BNRReminderViewController**



Além disso, o controlador de visão terá uma propriedade **datePicker** que aponta para o objeto **UIDatePicker**. Por fim, o controlador de visão será o alvo do **UIButton** e deve implementar seu método de ação **addReminder:**.

Como a view de **BNRReminderViewController** tem subvisões, será mais fácil criar a hierarquia de visão desse controlador de visão no Interface Builder.

## Criação de uma visão no Interface Builder

Primeiro, abra o `BNRReminderViewController.m`. Adicione uma extensão de classe para `BNRReminderViewController` que inclua uma declaração da propriedade `datePicker`. Depois adicione uma implementação simples para `addReminder`: que registre a data escolhida.

```
#import "BNRReminderViewController.h"

@interface BNRReminderViewController ()
@property (nonatomic, weak) IBOutlet UIDatePicker *datePicker;
@end

@implementation BNRReminderViewController

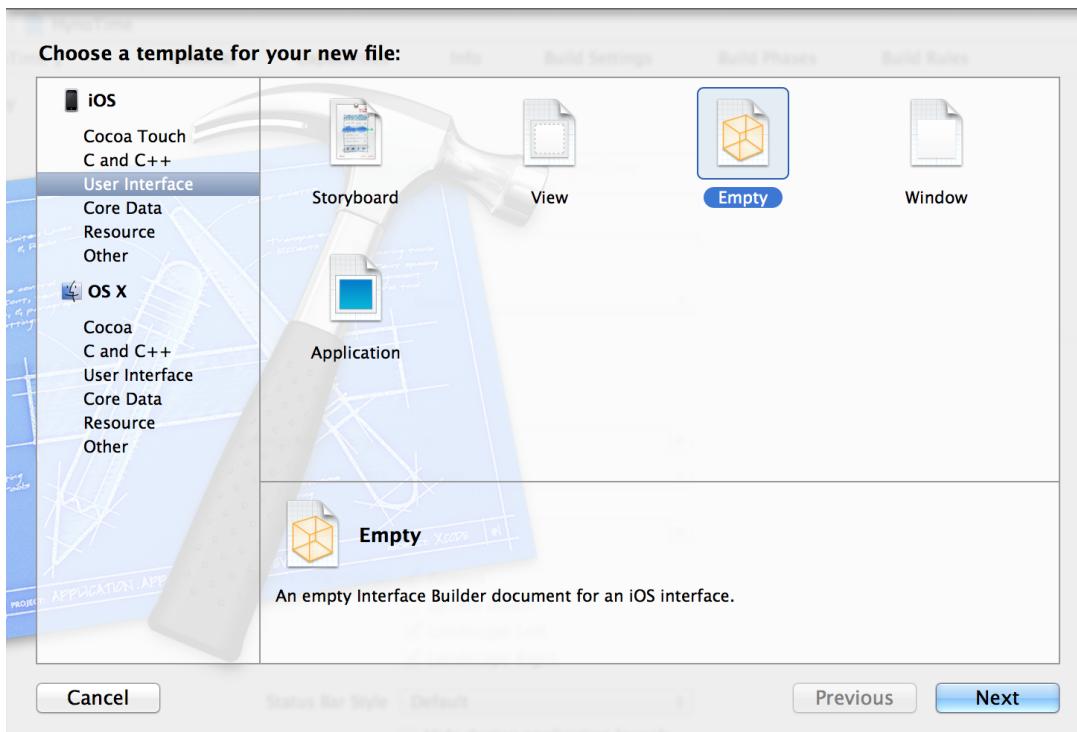
- (IBAction)addReminder:(id)sender
{
    NSDate *date = self.datePicker.date;
    NSLog(@"Setting a reminder for %@", date);
}

@end
```

Relembrando o Chapter 1: as palavras-chave `IBOutlet` e `IBAction` dizem ao Xcode que você fará essas conexões no Interface Builder. O primeiro passo é criar um arquivo XIB.

Crie um novo arquivo XIB selecionando File → New → File.... Na seção iOS, selecione User Interface, escolha o template Empty e clique em Next (Figure 6.8).

Figure 6.8 Criação de um XIB vazio



Selecione iPhone no menu pop-up que aparece e clique em Next.

Nomeie esse arquivo `BNRReminderViewController.xib` e salve-o. (É importante nomear esse e outros arquivos como estamos indicando. Algumas pessoas gostam de nomear os arquivos de forma diferente conforme fazem

os exercícios deste livro. Essa não é uma boa ideia. Muitos dos nomes são baseados em princípios embutidos no SDK do iOS.)

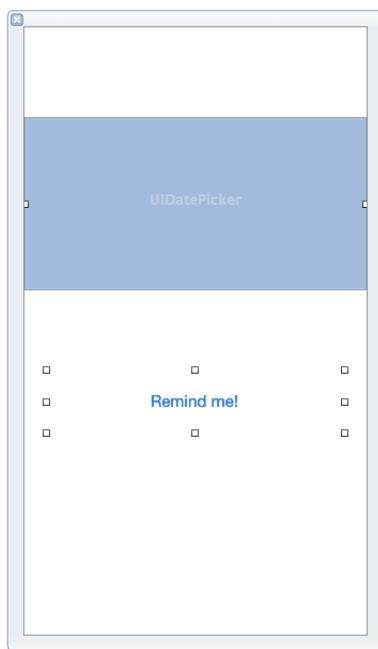
Agora, você tem um novo arquivo: `BNRReminderViewController.xib`. Selecione esse arquivo no navegador de projetos para abri-lo no Interface Builder.

## Criação de objetos de visão

Na biblioteca de objetos (no parte de baixo do painel direito do Xcode), procure por `UIView`. Arraste um objeto View para o canvas. Como padrão, ele terá o tamanho da tela, que é o que você quer.

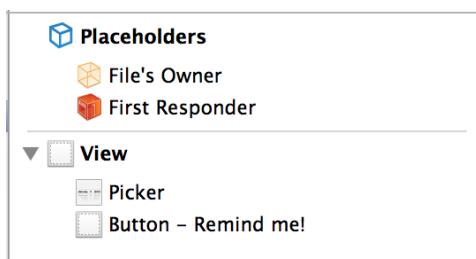
Em seguida, encontre um Date Picker e um Button na biblioteca e arraste-os para a visão. Posicione e redimensione as subvisões conforme mostrado na Figure 6.9. Lembre-se de que você pode clicar duas vezes sobre o botão para trocar seu título.

Figure 6.9 Arquivo XIB de `BNRReminderViewController`



Na estrutura do documento à esquerda do canvas, você pode selecionar sua hierarquia de visão: a View é a raiz, e Picker e Button são suas subvisões.

Figure 6.10 Hierarquia no `BNRReminderViewController.xib`



## Carregamento de um arquivo NIB

Quando um controlador de visão obtém sua hierarquia de visão carregando um arquivo NIB, você não sobrescreve `loadView`. A implementação-padrão de `loadView` sabe como lidar com o carregamento de um arquivo NIB.

A classe `BNRReminderViewController` precisa saber que arquivo NIB carregar. Você pode fazer isso no inicializador designado de `UIViewController`:

```
- (instancetype)initWithNibName:(NSString *)NibName bundle:(NSBundle *)nibBundle;
```

Nesse método, você passa o nome do arquivo NIB a ser carregado e o pacote no qual procurar esse arquivo.

No `BNRAppDelegate.m`, importe o `BNRReminderViewController.h`. Em seguida, crie uma instância de `BNRReminderViewController` e diga a ela onde encontrar seu arquivo NIB. Por fim, torne o objeto `BNRReminderViewController` o `rootViewController` da janela.

```
#import "BNRReminderViewController.h"

@implementation BNRAppDelegate

- (BOOL)application:(UIApplication *)application
    didFinishLaunchingWithOptions:(NSDictionary *)launchOptions
{
    self.window = [[UIWindow alloc] initWithFrame:[[UIScreen mainScreen] bounds]];
    // Override point for customization after application launch

    // This line will generate a warning, ignore it for now
    BNRHypnosisViewController *hvc = [[BNRHypnosisViewController alloc] init];

    // This will get a pointer to an object that represents the app bundle
    NSBundle *appBundle = [NSBundle mainBundle];

    // Look in the appBundle for the file BNRReminderViewController.xib
    BNRReminderViewController *rvc =
        [[BNRReminderViewController alloc] initWithNibName:@"BNRReminderViewController"
                                                bundle:appBundle];

    self.window.rootViewController = hvc;
    self.window.rootViewController = rvc;

    self.window.backgroundColor = [UIColor whiteColor];
    [self.window makeKeyAndVisible];
    return YES;
}
```

O pacote que você recebe ao enviar a mensagem `mainBundle` é o pacote do aplicativo. Esse pacote é um diretório do sistema de arquivos que contém o executável do arquivo e também os recursos (como arquivos NIB) que o executável usará. É aqui que ficará o `BNRReminderViewController.xib`.

Você já criou e configurou os objetos na hierarquia de visão. Já construiu um inicializador para o controlador de visão, para que ele possa encontrar e carregar o arquivo NIB correto. Você configurou o controlador de visão para ser o controlador de visão raiz para adicioná-lo à hierarquia de visão da janela. Mas, se você compilar e executar agora, o aplicativo irá falhar. Experimente e veja. Quando o aplicativo travar, observe a exceção no console:

```
'-[UIViewController _loadViewFromNibNamed:bundle:] loaded the
"BNRReminderViewController" nib but the view outlet was not set.'
```

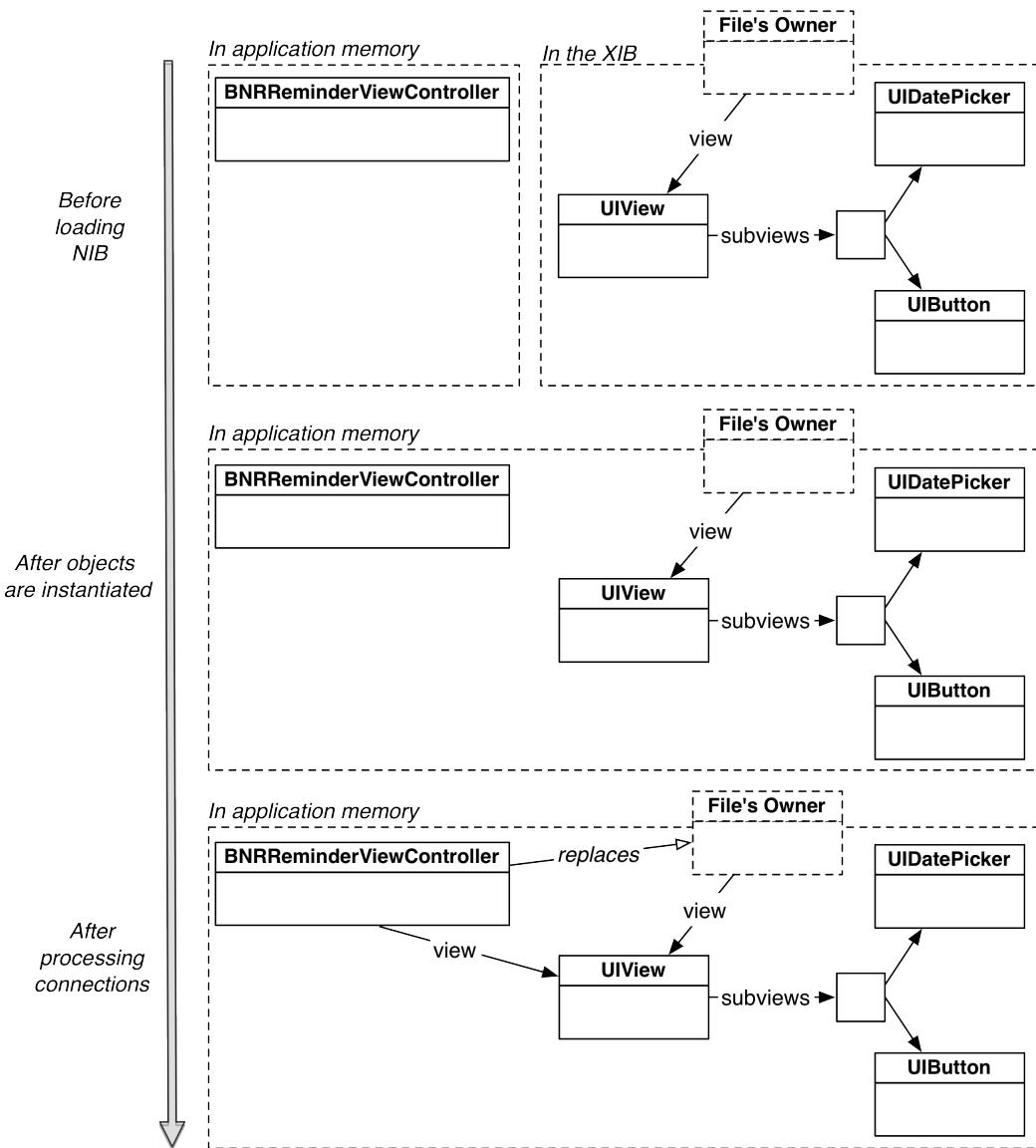
Quando o arquivo NIB correspondente foi carregado, esses objetos foram instanciados. Mas você não fez as conexões para vincular os objetos instanciados a `BNRReminderViewController` no aplicativo em execução. Isso inclui a propriedade `view` do controlador de visão. Assim, quando o controlador de visão tenta fazer com que sua `view` seja adicionada à tela, uma exceção é lançada, porque `view` é `nil`.

Como você pode associar um objeto de visão criado em um arquivo XIB com um controlador de visão em um aplicativo em execução? É aqui que o objeto File's Owner entra.

## Conexão a File's Owner

O objeto File's Owner é um placeholder – trata-se de um buraco deixado intencionalmente no arquivo XIB. Sendo assim, carregar um NIB é um processo que tem duas partes: instanciar todos os objetos arquivados no XIB e, depois, largar o objeto que está carregando o NIB no buraco File's Owner e estabelecer as conexões preparadas (Figure 6.11).

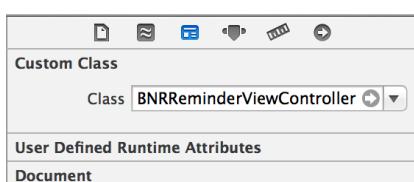
Figure 6.11 Linha do tempo do carregamento do NIB



Então, se você quiser conectar ao objeto que carrega o NIB no tempo de execução, você conecta a File's Owner quando estiver trabalhando no XIB. O primeiro passo é dizer ao arquivo XIB que o File's Owner será uma instância de **BNRReminderViewController**.

Reabra o `BNRReminderViewController.xib`. Selecione o objeto File's Owner na estrutura do documento. Em seguida, clique na guia na área do inspetor para mostrar o *inspetor de identidade*. Altere a Class do File's Owner para **BNRReminderViewController** (Figure 6.12).

Figure 6.12 Inspetor de identidade do File's Owner



Agora, você pode fazer as conexões que estão faltando. Vamos começar com o outlet `view`. Mantenha Control pressionado e clique no objeto File's Owner para exibir um painel de conexões disponíveis. Como a classe que

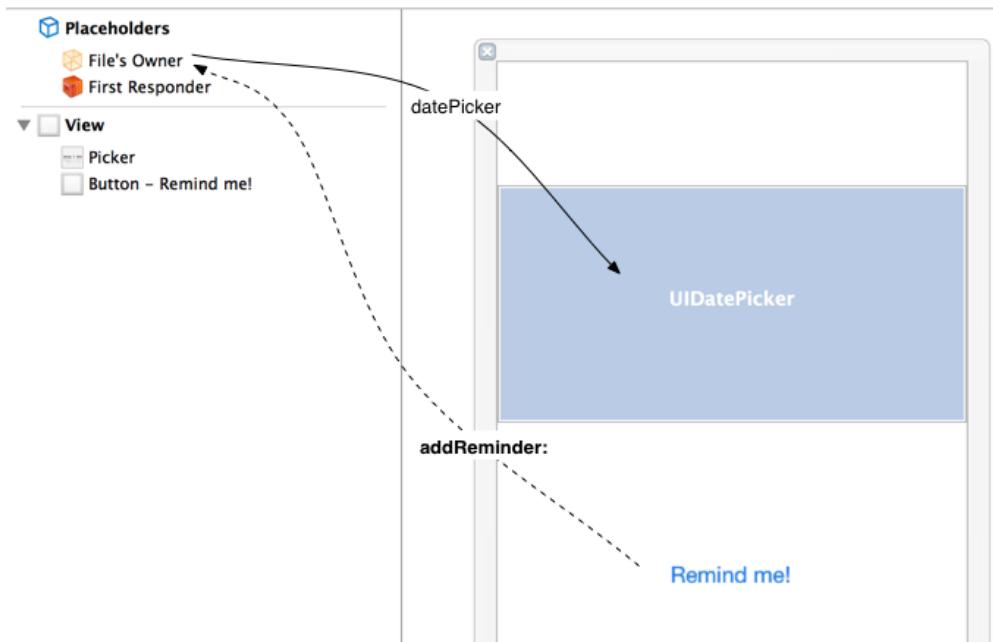
o File's Owner está representando é uma subclasse **UIViewController**, é oferecido a você sua propriedade **view** como um outlet (saída).

No painel de conexões disponíveis, arraste de **view** para o objeto **UIView** no canvas para configurar o outlet **view** para apontar para **UIView**.

Agora, quando **BNRReminderViewController** carregar o arquivo NIB, ela será capaz de carregar sua **view**. Compile e execute o aplicativo para confirmar que a **view** de **BNRReminderViewController** que você criou no **BNRReminderViewController.xib** agora aparece e que o aplicativo já não falha mais imediatamente.

Conclua fazendo as conexões restantes (Figure 6.13). Clique com o botão direito para revelar outlets do File's Owner e arraste para conectar o outlet **datePicker** a **UIDatePicker**. Em seguida, mantenha Control pressionado e arraste de **UIButton** no canvas para o File's Owner e selecione **addReminder:** para definir a ação.

Figure 6.13 Conexões XIB de **BNRReminderViewController**



Compile e execute o aplicativo. Selecione uma hora, toque no botão **Remind Me** e verifique a data do seu lembrete no console. Mais para a frente no capítulo, você atualizará o **addReminder:** para registrar uma notificação local.

Anteriormente, no **BNRReminderViewController.m**, você declarou o outlet **datePicker** como **weak** (fraco). Declarar outlets como **weak** (fracos) é uma convenção das versões mais antigas do iOS. Nessas versões, a **view** de um controlador de visão era automaticamente destruída toda vez que a memória do sistema estava baixa e era recriada posteriormente se necessário. Garantir que o controlador de visão tivesse apenas propriedades fracas de subvisões significava que destruir a **view** também destruía todas as suas subvisões e evitava vazamentos de memória.

## UITabBarController

Controladores de visão se tornam mais interessantes quando as ações do usuário podem fazer com que outro controlador de visão seja apresentado. Neste livro, você aprenderá diferentes maneiras de apresentar controladores de visão. Você começará com uma **UITabBarController** que permitirá ao usuário alternar entre instâncias de **BNRHypnosisViewController** e **BNRReminderViewController**.

**UITabBarController** mantém um array de controladores de visão. Ela mantém também uma barra de guias na parte inferior da tela com uma guia para cada controlador de visão nesse array. Tocar em uma guia faz com que a visão do controlador de visão associado a essa guia seja apresentada.

No **BNRAppDelegate.m**, crie uma instância de **UITabBarController**, dê a ela ambos os controladores de visão e instale-a como **rootViewController** da janela.

```
- (BOOL)application:(UIApplication *)application
    didFinishLaunchingWithOptions:(NSDictionary *)launchOptions
{
    self.window = [[UIWindow alloc] initWithFrame:[[UIScreen mainScreen] bounds]];
    // Override point for customization after application launch

    BNRHypnosisViewController *hvc = [[BNRHypnosisViewController alloc] init];
    // This will get a pointer to an object that represents the app bundle
    NSBundle *appBundle = [NSBundle mainBundle];

    // Look in the AppBundle for the file BNRReminderViewController.xib
    BNRReminderViewController *rvc = [[BNRReminderViewController alloc]
        initWithNibName:@"BNRReminderViewController"
        bundle:appBundle];

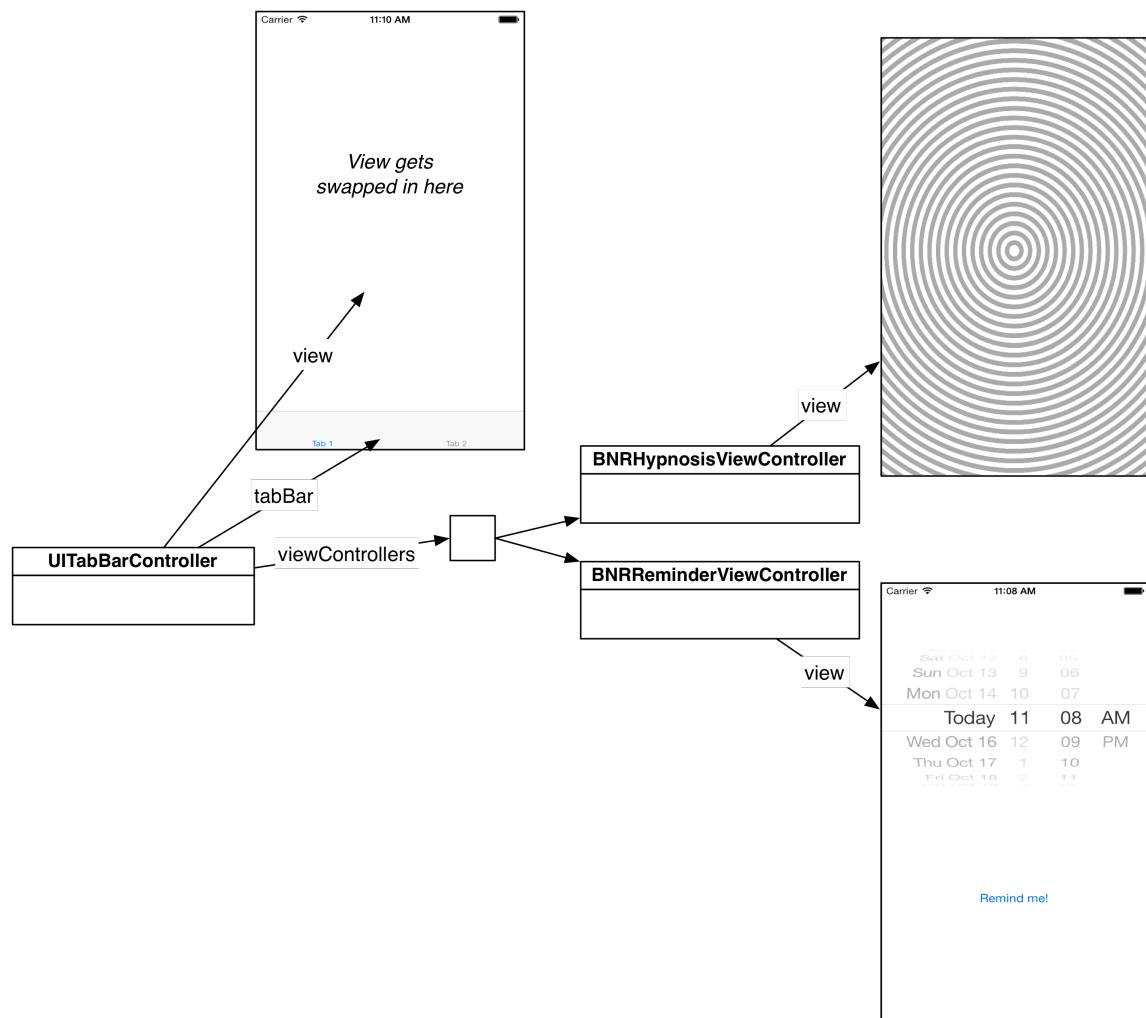
    UITabBarController *tabBarController = [[UITabBarController alloc] init];
    tabBarController.viewControllers = @[hvc, rvc];

    self.window.rootViewController = rvc;
    self.window.rootViewController = tabBarController;

    self.window.backgroundColor = [UIColor whiteColor];
    [self.window makeKeyAndVisible];
    return YES;
}
```

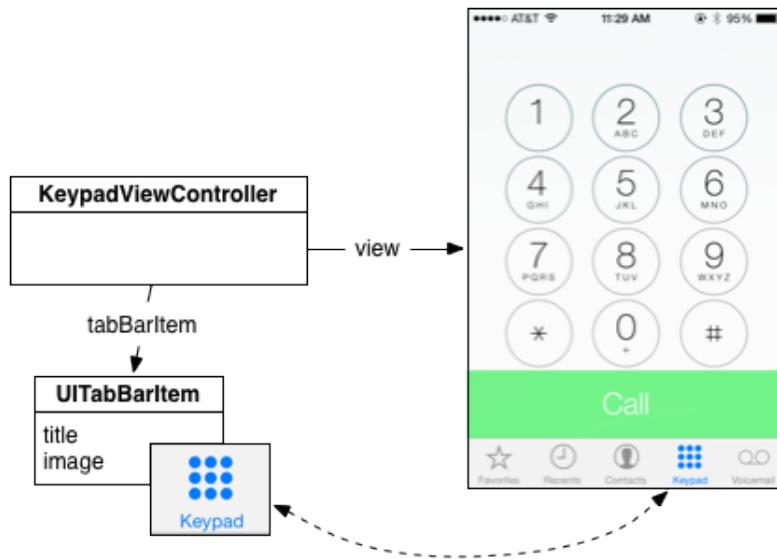
Compile e execute o aplicativo. A barra na parte inferior trata-se, na verdade, de duas guias. Toque nos lados esquerdo e direito da barra de guias para alternar entre os dois controladores de visão. Na próxima seção, você criará itens da barra de guias para tornar as duas guias óbvias.

**UITabBarController** é, por si só, uma subclasse de **UIViewController**. Uma view de **UITabBarController** é uma **UIView** com duas subvisões: a barra de guias e a view do controlador de visão selecionado (Figure 6.14).

Figure 6.14 Diagrama de **UITabBarController**

## Itens da barra de guias

Cada guia na barra de guias pode exibir um título e uma imagem. Cada controlador de visão mantém uma propriedade `tabBarItem` para essa finalidade. Quando um controlador de visão está contido em `UITabBarController`, seu item da barra de guias aparece na barra de guias. A Figure 6.15 mostra um exemplo desse relacionamento no aplicativo Phone do iPhone.

Figure 6.15 Exemplo de **UITabBarItem**

Primeiro, você precisa adicionar alguns arquivos a seu projeto que serão as imagens dos itens da barra de guias. Abra o Asset Catalog abrindo o `Images.xcassets` no navegador de projetos. Em seguida, encontre `Hypno.png`, `Time.png`, `Hypno@2x.png` e `Time@2x.png` no diretório Resources do arquivo que você baixou anteriormente (<http://www.bignerdranch.com/solutions/iOSProgramming4ed.zip>). Arraste esses arquivos para dentro da lista de conjunto de imagens no lado esquerdo do Asset Catalog.

No `BNRHypnosisViewController.m`, sobrescreva o inicializador designado de `UIViewController`, `initWithNibName:bundle:`, para obter e definir um item da barra de guias para `BNRHypnosisViewController`.

```

- (instancetype)initWithNibName:(NSString *)NibNameOrNil
                           bundle:(NSBundle *)bundleOrNil
{
    self = [super initWithNibName:nibNameOrNilOrNil
                           bundle:nibBundleOrNilOrNil];
    if (self) {
        // Set the tab bar item's title
        self.tabBarItem.title = @"Hypnotize";

        // Create a UIImage from a file
        // This will use Hypno@2x.png on retina display devices
        UIImage *i = [UIImage imageNamed:@"Hypno.png"];

        // Put that image on the tab bar item
        self.tabBarItem.image = i;
    }
    return self;
}
    
```

No `BNRReminderViewController.m`, faça a mesma coisa.

```

- (instancetype)initWithNibName:(NSString *)NibNameOrNil
                           bundle:(NSBundle *)bundleOrNil
{
    self = [super initWithNibName:NibNameOrNil
                           bundle:bundleOrNil];
    if (self) {
        // Get the tab bar item
        UITabBarItem *tbi = self.tabBarItem;

        // Give it a label
        tbi.title = @"Reminder";

        // Give it an image
        UIImage *i = [UIImage imageNamed:@"Time.png"];
        tbi.image = i;
    }

    return self;
}

```

Compile e execute o aplicativo, e você verá úteis imagens e títulos na barra de guias. (Figure 6.16).

Figure 6.16 Itens da barra de guias com rótulos e ícones



## Inicializadores do UIViewController

Quando você criou um item da barra de guias para **BNRHypnosisViewController**, sobrescreveu o **initWithNibName:bundle:**. No entanto, quando você inicializou a instância **BNRHypnosisViewController** no **BNRAppDelegate.m**, enviou a ela **init** e continuou os com itens da barra de guias. Isso aconteceu porque **initWithNibName:bundle:** é o inicializador designado de **UIViewController**. Enviar **init** a um controlador de visão chama **initWithNibName:bundle:** e passa nil para ambos os argumentos.

**BNRHypnosisViewController** não usa um arquivo NIB para criar sua visão, portanto, o parâmetro do nome de arquivo é irrelevante. O que acontece se você enviar **init** a um controlador de visão que usa de fato um arquivo NIB? Vamos descobrir.

No **BNRAppDelegate.m**, altere seu código para inicializar **BNRReminderViewController** com **init**, em vez de **initWithNibName:bundle:**.

```

BNRHypnosisViewController *hvc = [[BNRHypnosisViewController alloc] init];

// This will get a pointer to an object that represents the app bundle
NSBundle *appBundle = [NSBundle mainBundle];

// Look in the AppBundle for the file BNRReminderViewController.xib
BNRReminderViewController *rvc =
    [[BNRReminderViewController alloc] initWithNibName:@"BNRReminderViewController"
                                              bundle:appBundle];

BNRReminderViewController *rvc = [[BNRReminderViewController alloc] init];
UITabBarController *tabBarController = [[UITabBarController alloc] init];

```

Compile e execute o aplicativo, e ele funcionará exatamente como antes. Quando um controlador de visão é inicializado com nil como seu nome NIB, ele procura um arquivo NIB com o mesmo nome da classe. Passar nil como o pacote significa que o controlador de visão procurará no pacote principal do aplicativo. Assim, **BNRReminderViewController** ainda procurará o **BNRReminderViewController.xib** no pacote principal.

É por isso que avisamos a você antes para se atter aos nomes dados ao nomear arquivos. Se você estiver criando uma classe **FidoViewController** que busque sua visão em um arquivo NIB, o único nome adequado para esse arquivo XIB é **FidoViewController.xib**.

## Adição de uma notificação local

Agora, você irá implementar a função de lembrete usando uma *notificação local*. Uma notificação local é uma maneira de um aplicativo alertar o usuário mesmo quando o aplicativo não estiver sendo executado no momento.

(Um aplicativo também pode usar notificações push que são implementadas usando um servidor back-end. Para saber mais a respeito de notificações push, leia o *Local and Push Notification Programming Guide* da Apple.)

Fazer com que uma notificação seja exibida é fácil. Você cria uma **UILocalNotification** e dá a ela algum texto e uma data. Em seguida, você programa a notificação com o aplicativo compartilhado – a única instância de **UIApplication**.

Atualize o método **addReminder:** para fazer isso:

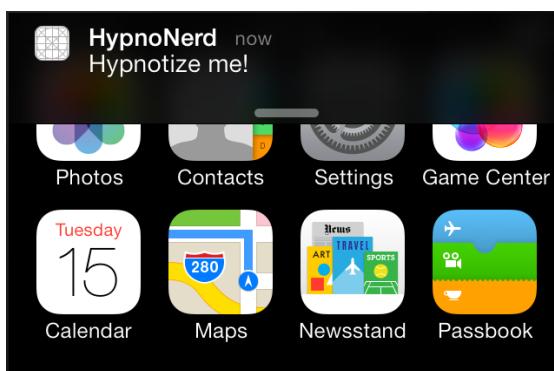
```
- (IBAction)addReminder:(id)sender
{
    NSDate *date = self.datePicker.date;
    NSLog(@"Setting a reminder for %@", date);

    UILocalNotification *note = [[UILocalNotification alloc] init];
    note.alertBody = @"Hypnotize me!";
    note.fireDate = date;

    [[UIApplication sharedApplication] scheduleLocalNotification:note];
}
```

Compile e execute o aplicativo. Use o seletor de data para selecionar uma hora bem próxima e toque no botão Remind Me. Para ver a notificação, o HypnoNerd não pode estar em primeiro plano. Pressione o botão Home na parte inferior do dispositivo ou selecione Hardware → Home no simulador. Quando chegar a hora que você escolheu, um banner de notificação aparecerá no topo da tela (Figure 6.17). Tocar rapidamente na notificação executará o aplicativo HypnoNerd.

Figure 6.17 Notificação local



Há um problema: o usuário pode selecionar uma hora no passado. Seria bom se o seletor de datas não permitisse isso. Você logo cuidará disso.

## Visões carregadas e exibidas

Agora que você tem dois controladores de visão, o carregamento lento de visões sobre o qual você aprendeu se torna mais importante.

Quando o aplicativo é iniciado, o controlador da barra de guias assume o padrão de carregar a visão do primeiro controlador de visão em seu array, **BNRHypnosisViewController**. Isso significa que a visão de **BNRReminderViewController** não é necessária e só será necessária quando (ou se) o usuário tocar na guia paravê-la.

Você mesmo pode testar esse comportamento – quando um controlador de visão terminar de carregar sua visão, ele recebe a mensagem **viewDidLoad**.

No **BNRHypnosisViewController.m**, sobrescreva **viewDidLoad** para registrar uma instrução no console.

```
- (void)viewDidLoad
{
    // Always call the super implementation of viewDidLoad
    [super viewDidLoad];

    NSLog(@"BNRHypnosisViewController loaded its view.");
}
```

No BNRReminderViewController.m, sobrescreva o mesmo método.

```
- (void)viewDidLoad
{
    [super viewDidLoad];

    NSLog(@"BNRReminderViewController loaded its view.");
}
```

Compile e execute o aplicativo. O console relata que **BNRHypnosisViewController** carregou sua view imediatamente. Toque na guia de **BNRReminderViewController**, e o console reportará que sua view agora está carregada. Nesse ponto, ambas as visões foram carregadas, então alternar entre as guias não irá mais disparar o método **viewDidLoad**. (Experimente e veja.)

Para preservar os benefícios do carregamento lento, você jamais deve acessar a propriedade **view** de um controlador de visão no **initWithNibName:bundle:**. Solicitar a **view** no inicializador fará com que o controlador de visão carregue sua **view** prematuramente.

## Acesso a subvisões

Com frequência, você irá querer fazer uma inicialização extra das subvisões que estão definidas no arquivo XIB antes de elas aparecerem para o usuário. Entretanto, você não pode fazer isso no inicializador desse controlador de visão porque o arquivo NIB ainda não foi carregado. Se você tentar, todos os ponteiros que o controlador de visão declarar que, eventualmente, apontará para subvisões estarão apontando para **nil**. O compilador não reclamará se você enviar uma mensagem para um desses ponteiros, mas nada do que você tenha pretendido que acontecesse com aquele objeto de visão acontecerá.

Sendo assim, onde você pode acessar uma subvisão? Há duas opções principais, dependendo do que você precisar fazer. A primeira opção é o método **viewDidLoad** que você sobrescreveu para identificar o carregamento lento. O controlador de visão recebe essa mensagem após o arquivo NIB do controlador de visão ser carregado, momento no qual todos os ponteiros do controlador de visão estarão apontando para os objetos adequados. A segunda opção é outro método **UIViewController viewWillAppear:**. O controlador de visão recebe essa mensagem logo antes de sua **view** ser adicionada à janela.

Qual é a diferença? Você sobrescreve **viewDidLoad** se a configuração somente precisar ser feita uma vez durante a execução do aplicativo. Você sobrescreve **viewWillAppear:** se precisar que a configuração seja feita e refeita toda vez que o controlador de visão aparecer na tela.

Há uma subvisão da visão de **BNRReminderViewController** que precisa de um trabalho extra – o seletor de data. No momento, os usuários podem escolher horários de lembrete no passado. Você irá configurar o seletor de data para permitir que os usuários selecionem apenas um horário que seja ao menos 60 segundos no futuro.

Isso é algo que precisará ser feito toda vez que a visão aparecer, não apenas uma vez depois de a visão ser carregada; por isso, você irá sobrescrever **viewWillAppear:**.

No BNRReminderViewController.m, sobrescreva **viewWillAppear:** para configurar a **minimumDate** do seletor de data.

```
- (void)viewWillAppear:(BOOL)animated
{
    [super viewWillAppear:animated];
    self.datePicker.minimumDate = [NSDate dateWithTimeIntervalSinceNow:60];
}
```

Compile e execute o aplicativo. Selecione a guia Reminder e confirme que o seletor de data só permitirá ao usuário selecionar uma data no futuro.

Se você tivesse sobreescrito `viewDidLoad` em vez disso, a `minimumDate` de `datePicker` seria configurada para 60 segundos após a visão ter sido inicialmente carregada e, provavelmente, permaneceria inalterada durante toda a execução do aplicativo. Se o aplicativo fosse executado durante muito tempo, os usuários logo poderiam escolher horários no passado. Às vezes, a view de um controlador de visão pode ser destruída e recarregada, mas esse não é o comportamento típico em dispositivos mais novos.

Está pensando no flag animado deste método? Ele indica se a transição de aparecimento e desaparecimento é animada ou não. No caso de `UITabBarController`, a transição não é animada. Mais adiante neste livro, no Chapter 10, você usará `UINavigationController`, que anima controladores de visão que estão sendo empurrados para dentro e para fora da tela.

## Interação com controladores de visão e suas visões

Vamos dar uma olhada em alguns métodos que são chamados durante o ciclo de vida de um controlador de visão e sua visão. Você já viu alguns desses métodos, outros são novos:

- `application:didFinishLaunchingWithOptions:`: é onde você instancia e define o controlador de visão raiz de um aplicativo.

Esse método é chamado exatamente uma vez quando o aplicativo é iniciado. Mesmo se você for para outro aplicativo e retornar, esse método não será chamado novamente. Se você reiniciar seu telefone e iniciar o aplicativo novamente, `application:didFinishLaunchingWithOptions:` será chamado novamente.

- `initWithNibName:bundle:`: é o inicializador designado para `UIViewController`.

Quando uma instância do controlador de visão é criada, seu `initWithNibName:bundle:` é chamado uma vez. Observe que, em alguns aplicativos, você pode terminar criando diversas instâncias da mesma classe de controlador de visão. Esse método será chamado uma vez em cada uma, à medida que elas são criadas.

- `loadView`: é sobreescrito para criar uma visão do controlador de visão de forma programática.
- `viewDidLoad` pode ser sobreescrito para configurar visões criadas pelo carregamento de um arquivo NIB. Esse método é chamado após a visão de um controlador de visão ser criada.
- `viewWillAppear`: pode ser sobreescrito para configurar visões criadas pelo carregamento de um arquivo NIB.

Esse método e `viewDidAppear`: serão chamados toda vez que seu controlador de visão for movido na tela. `viewWillDisappear`: e `viewDidDisappear`: serão chamados toda vez que seu controlador de visão for movido para fora da tela. Então, se você iniciar o aplicativo no qual está trabalhando e alternar entre Hypnosis e Reminder, o método `viewDidLoad` de `BNRReminderViewController` será chamado uma vez, mas `viewWillAppear`: será chamado dezenas de vezes.

## Desafio de bronze: outra guia

Dê a `UITabBarController` uma terceira guia que apresente um quiz ao usuário. (Dica: você pode reutilizar arquivos do seu projeto Quiz para este desafio.)

## Desafio de prata: lógica de controlador

Adicione `UISegmentedControl` à view de `BNRHypnosisViewController` com segmentos para Red, Green e Blue. Quando o usuário tocar no controle segmentado, altere a cor dos círculos na `BNRHypnosisView`. Certifique-se de criar uma cópia do projeto e trabalhe com a cópia durante este exercício.

## Para os mais curiosos: codificação de chave-valor

Quando um arquivo NIB é lido, os outlets são definidos com o uso de um mecanismo chamado *codificação de chave-valor* (Key-value coding, ou KVC). Codificação de chave-valor é um conjunto de métodos definidos em `NSObject` que permite a você definir e obter os valores de propriedades por nome. Aqui estão dois dos métodos:

- `(id)valueForKey:(NSString *)k;`

```
- (void)setValue:(id)v forKey:(NSString *)k;
```

**valueForKey:** é um método getter universal. Você pode perguntar a qualquer objeto o valor de sua propriedade **fido** assim:

```
id currentFido = [selectedObj valueForKey:@"fido"];
```

Se houver um método **fido** (o getter específico a **fido**), ele será chamado, e o valor retornado será usado. Se não houver nenhum método **fido**, o sistema procurará uma variável de instância chamada **\_fido** ou **fido**. Se existir qualquer uma das variáveis de instância, o valor da variável de instância será usado. Se não existir um acessor nem uma variável de instância, uma exceção será lançada.

**setValue:forKey:** é um método setter universal. Ele permite a você configurar o valor da propriedade **fido** de um objeto assim:

```
[selectedObject setValue:userChoice forKey:@"fido"];
```

Se houver um método **setFido:**, ele será chamado. Se não houver nenhum método assim, o sistema procurará uma variável chamada **\_fido** ou **fido** e definirá o valor dessa variável diretamente. Se não existir um acessor nem uma das variáveis de instância, uma exceção será lançada.

Quando o arquivo NIB está sendo carregado, os outlets são definidos usando **setValue:forKey:**. Desse modo, se você definir um outlet **rex** para um objeto no Interface Builder, esse objeto deve ter um acessor chamado **setRex:**, uma variável de instância chamada **rex**, ou uma variável de instância chamada **\_rex**. Se você não tiver nenhum destes, uma exceção será lançada quando o arquivo NIB for lido no tempo de execução. O erro será semelhante ao seguinte:

```
[<BNRSunsetViewController 0x68c0740> setValue:forUndefinedKey:]:
this class is not key value coding-compliant for the key rex.'
```

Tipicamente, um desenvolvedor vê esse erro quando cria uma propriedade de outlet, conecta-a no Interface Builder e, depois, renomeia a propriedade.

A lição mais importante desta seção: usar as convenções de nomenclatura de método acessor é mais do que apenas uma coisa legal que você faz por outras pessoas que possam ler seu código. O sistema espera que um método chamado **setFido:** seja o setter para a propriedade **fido**. O sistema espera que um método chamado **fido** seja o getter para a propriedade **fido**. Coisas ruins acontecem quando você viola as convenções de nomenclatura.

Vou lhe dar um exemplo. Quando eu era mais jovem, criei uma classe de controlador com um outlet chamado **clock** que apontava para uma visão em estilo relógio. Eu também tinha um botão que acionava um método de ação (no mesmo controlador) que ia à Internet, obtinha a hora correta e atualizava a visão em estilo relógio. Eu, desentendido que era, nomeei esse método de ação da seguinte forma:

```
- (IBAction)setClock:(id)sender;
```

Isso gerou um bug estranhíssimo: Quando o arquivo NIB foi carregado, o método de ação foi acionado imediatamente. E o outlet **clock** nunca era definido devidamente, apesar de eu tê-lo conectado corretamente no arquivo NIB. Por quê? O sistema estava tentando usar meu método de ação **setClock:** como se fosse um acessor para configurar meu outlet **clock**.

Renomeei o método para **updateClock:**, e tudo funcionou perfeitamente – mas foram quatro horas da minha vida que nunca recuperarei. Seguir convenções de nomenclatura é muito importante para desenvolvedores de iOS.

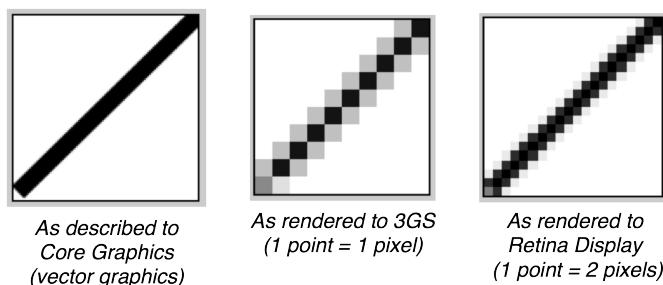
## Para os mais curiosos: tela Retina

Com o lançamento do iPhone 4, a Apple introduziu a tela Retina para o iPhone e o iPod touch. A tela Retina tem uma resolução muito alta – 640x1136 pixels (em uma tela de 4 polegadas) e 640x960 pixels (em uma tela de 3,5 polegadas) em comparação com os 320x480 pixels de dispositivos anteriores. Vejamos o que você precisa fazer para maximizar a qualidade dos gráficos em ambas as telas.

Para gráficos vetoriais, como o método **drawRect:** de **BNRHypnosisView** e texto desenhado, você não precisa fazer nada; o mesmo código será renderizado com a qualidade máxima permitida pelo dispositivo. No entanto, se você desenhar usando as funções do Core Graphics, a aparência desses gráficos irá variar nos diferentes

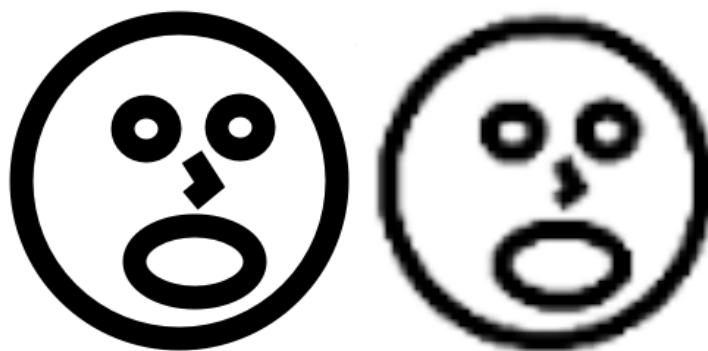
dispositivos. No Core Graphics, também chamado de Quartz, linhas, curvas, texto etc. são descritos em termos de *pontos*. Em uma tela comum, um ponto é 1x1 pixel. Em uma tela Retina, um ponto são 2x2 pixels (Figure 6.18).

Figure 6.18 Renderização para resoluções diferentes



Dadas essas diferenças, imagens bitmap (como arquivos JPEG ou PNG) não ficarão atraentes se a imagem não for personalizada para o tipo de tela do dispositivo. Digamos que o seu aplicativo inclua uma pequena imagem de 25x25 pixels. Se essa imagem será exibida em uma tela Retina, ela deverá ser ampliada para cobrir uma área de 50x50 pixels. Nesse momento, o sistema faz um tipo de ponderação, chamada antisserrilhado (anti-aliasing), para evitar um efeito serrilhado na imagem. O resultado é uma imagem sem serrilhado – mas também sem foco (Figure 6.19).

Figure 6.19 Falta de foco devido à ampliação da imagem



Você pode alternativamente usar um arquivo maior, mas a ponderação nesse caso causaria problemas na outra direção, quando a imagem é reduzida para uma tela comum. A única solução é empacotar dois arquivos de imagem junto com o seu aplicativo: um com resolução de pixels igual ao número de pontos na tela para as telas comuns e outro com o dobro desse tamanho em pixels para telas Retina.

Felizmente, você não precisa escrever nenhum código adicional para determinar qual imagem será carregada em qual dispositivo. Você só precisa sufixar a imagem de resolução mais alta com @2x. Então, quando você usa o método  `imageNamed:` de `UIImage` para carregar a imagem, esse método procura no pacote e obtém o arquivo que é adequado para o dispositivo específico.

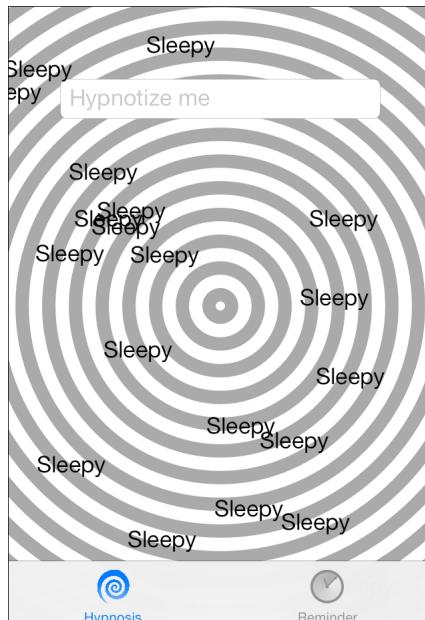


# Delegação e entrada de texto

Neste capítulo, apresentaremos a delegação, um padrão de projeto recorrente do desenvolvimento em Cocoa Touch. Além disso, você verá como usar o depurador que o Xcode oferece para encontrar e corrigir problemas no código.

No final do capítulo, o usuário do HypnoNerd poderá exibir mensagens hipnóticas na tela usando um campo de texto (Figure 7.1).

Figure 7.1 HypnoNerd concluído



## Campos de texto

Abra o aplicativo HypnoNerd que você iniciou no capítulo anterior.

Você já viu uma maneira de exibir texto em suas interfaces de usuário usando uma **UILabel**. Vamos ver agora uma outra maneira de exibir texto usando uma **UITextField**. Uma instância de **UITextField** permite que o usuário modifique o texto, muito semelhante a um campo de nome de usuário ou senha em um site.

Abra o `BNRHypnosisViewController.m` e modifique o `loadView` para adicionar uma **UITextField** a sua visão.

```
- (void)loadView
{
    CGRect frame = [UIScreen mainScreen].bounds;
    BNRHypnosisView *backgroundView = [[BNRHypnosisView alloc] initWithFrame:frame];

    CGRect textFieldRect = CGRectMake(40, 70, 240, 30);
    UITextField *textField = [[UITextField alloc] initWithFrame:textFieldRect];

    // Setting the border style on the text field will allow us to see it more easily
    textField.borderStyle = UITextBorderStyleRoundedRect;
    [backgroundView addSubview:textField];

    self.view = backgroundView;
}
```

Compile e execute o aplicativo, e você verá o campo de texto na guia Hypnotize. Toque no campo de texto; o teclado deslizará de baixo para cima da tela, permitindo que você insira texto. Para compreender como isso acontece detalhadamente, você precisa entender o *primeiro respondente*.

## UIResponder

**UIResponder** é uma classe abstrata no framework UIKit. Trata-se da superclasse de três classes que você já encontrou:

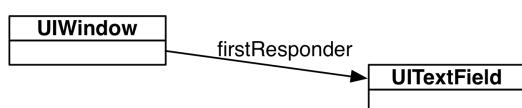
- **UIView**
- **UIViewController**
- **UIApplication**

**UIResponder** define métodos de tratamento de (ou “resposta a”) eventos: eventos de toque, eventos de movimentação (como uma sacudida) e eventos de controle remoto (como pausa ou reprodução). As subclasses sobrescrevem esses métodos para personalizar como respondem a eventos.

Em eventos de toque, é evidente a visão que o usuário tocou. Eventos de toque são enviados diretamente para essa visão. Você viu um exemplo disso no Chapter 5.

E os outros tipos de eventos? A **UIWindow** possui um ponteiro chamado **firstResponder**, que indica quem deve responder aos outros tipos de eventos. Quando você seleciona um campo de texto, por exemplo, a janela move o ponteiro de **firstResponder** para esse campo de texto. Eventos de movimentação e controle remoto são enviados para o primeiro respondente.

Figure 7.2 firstResponder



Quando um campo de texto ou uma visão de texto torna-se **firstResponder**, o teclado é exibido. Quando ele perde o status de primeiro respondente, o teclado é ocultado. Se você deseja que uma dessas visões se torne o primeiro respondente, envie a ela a mensagem **becomeFirstResponder**, e o teclado aparecerá. Quando quiser ocultar o teclado, envie a ele a mensagem **resignFirstResponder**.

A maioria das visões recusa-se a se tornar o primeiro respondente, elas não querem roubar o foco da visão do texto ou do campo de texto atualmente selecionado. Uma instância de **UISlider**, por exemplo, trata de eventos de toque, mas nunca aceitará o status de primeiro respondente.

## Configuração do teclado

A aparência do teclado é determinada por um conjunto de propriedades da **UITextField** chamado **UITextInputTraits**. Vamos modificar alguns deles para fornecer ao campo de texto algum texto de placeholder e para modificar o tipo de retorno do teclado.

```

- (void)loadView
{
    CGRect frame = [UIScreen mainScreen].bounds;
    BNRHypnosisView *backgroundView = [[BNRHypnosisView alloc] initWithFrame:frame];

    CGRect textFieldRect = CGRectMake(40, 70, 240, 30);
    UITextField *textField = [[UITextField alloc] initWithFrame:textFieldRect];

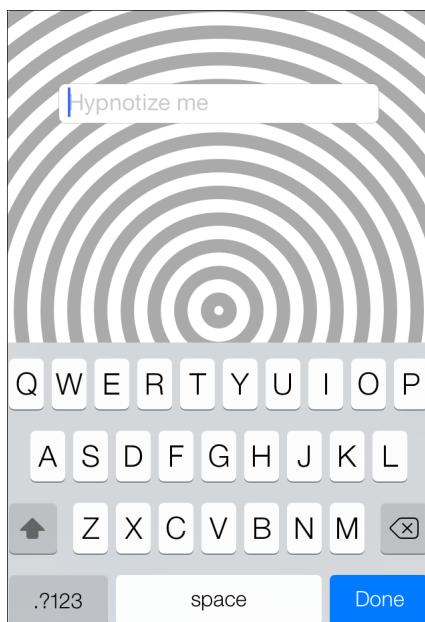
    // Setting the border style on the text field will allow us to see it more easily
    textField.borderStyle = UITextBorderStyleRoundedRect;
    textField.placeholder = @"Hypnotize me";
    textField.returnKeyType = UIReturnKeyTypeDone;

    [backgroundView addSubview:textField];
    self.view = backgroundView;
}

```

Compile e execute o aplicativo. Agora, o campo de texto possui uma string de placeholder, que será exibida até que o usuário digite algum texto. Além disso, a tecla de retorno agora informa Done, em vez do padrão Return. A Figure 7.3 mostra como fica a aparência da interface com essas alterações.

Figure 7.3 Campo de texto configurado



Se você tocar na tecla Done, perceberá que nada acontece. A alteração do tipo de tecla de retorno não causa impacto na funcionalidade da tecla de retorno. De fato, a tecla de retorno não faz nada automaticamente; você mesmo precisa implementar a funcionalidade da tecla de retorno. Antes disso, no entanto, vamos dar uma olhada em algumas outras propriedades úteis que podem ser usadas na configuração do teclado.

|                               |                                                                                                                                                                                                |
|-------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| autocapitalizationType        | Determina como serão tratadas as maiúsculas e minúsculas. As opções são nenhuma, palavras, sentenças ou todos os caracteres.                                                                   |
| autocorrectionType            | Fará sugestão e corrigirá palavras desconhecidas. Esse valor pode ser YES ou NO.                                                                                                               |
| enablesReturnKeyAutomatically | Esse valor pode ser YES ou NO. Se configurada como “yes” e nenhum texto for inserido, a tecla de retorno será desativada. Assim que qualquer texto é inserido, a tecla de retorno é reativada. |
| keyboardType                  | Determina o tipo de teclado que será exibido. Alguns exemplos são o teclado ASCII, o teclado de endereço de e-mail, o teclado numérico e o teclado de URL.                                     |

secureTextEntry

A configuração para YES faz com que o campo de texto se comporte como um campo de senha, ocultando o texto que é inserido.

## Delegação

Você já viu o padrão destino-ação. Ele é uma forma de callback usada pelo UIKit: Quando um botão é tocado, ele envia sua mensagem de ação para o destino. Geralmente, o código que você escreveu é disparado.

O comportamento de um botão é relativamente simples. Para objetos com comportamento mais complexo, como um campo de texto, a Apple usa o *padrão de delegação*. Você apresenta o campo de texto para um de seus objetos: “Este é seu delegate; quando alguma coisa interessante acontecer com você, envie uma mensagem para ele.” O campo de texto mantém um ponteiro para seu delegate. Muitas das mensagens que ele envia para seus delegates são informativas: “OK, terminei a edição!”. Veja algumas delas:

- `(void)textFieldDidEndEditing:(UITextField *)textField;`
- `(void)textFieldDidBeginEditing:(UITextField *)textField;`

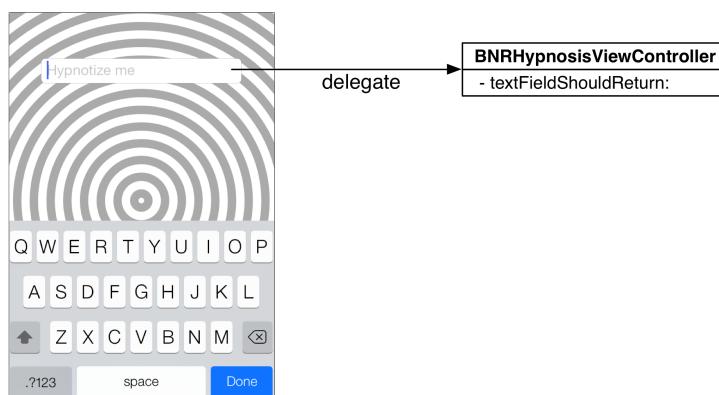
Observe que ele sempre envia a si próprio como o primeiro argumento para o método delegate.

Algumas das mensagens que ele envia para seus delegates são perguntas: “Eu estou finalizando a edição e ocultarei o teclado. Tudo bem?” Veja algumas delas:

- `(BOOL)textFieldShouldEndEditing:(UITextField *)textField;`
- `(BOOL)textFieldShouldBeginEditing:(UITextField *)textField;`
- `(BOOL)textFieldShouldClear:(UITextField *)textField;`
- `(BOOL)textFieldShouldReturn:(UITextField *)textField;`

Agora, você transformará sua **BNRHypnosisViewController** em delegate do campo de texto. Você implementará o método `textFieldShouldReturn:`. Ao executá-lo, você verá que o método é chamado automaticamente quando o usuário toca no botão Done.

Figure 7.4 **BNRHypnosisViewController** como delegate UITextField



No **BNRHypnosisViewController.m**, atualize o `loadView` para definir a propriedade `delegate` da **UITextField**, para que aponte para **BNRHypnosisViewController**.

```

- (void)loadView
{
    CGRect frame = [UIScreen mainScreen].bounds;
    BNRHypnosisView *backgroundView = [[BNRHypnosisView alloc] initWithFrame:frame];

    CGRect textFieldRect = CGRectMake(40, 70, 240, 30);
    UITextField *textField = [[UITextField alloc] initWithFrame:textFieldRect];

    // Setting the border style on the text field will allow us to see it more easily
    textField.borderStyle = UITextBorderStyleRoundedRect;
    textField.placeholder = @"Hypnotize me";
    textField.returnKeyType = UIReturnKeyTypeDone;

    // There will be a warning on this line. We will discuss it shortly.
    textField.delegate = self;

    [backgroundView addSubview:textField];
    self.view = backgroundView;
}

```

O método **textFieldShouldReturn:** utiliza apenas um argumento: o campo de texto cuja tecla de retorno foi tocada. Por enquanto, o aplicativo exibirá apenas o texto do campo de texto no console.

No `BNRHypnosisViewController.m`, implemente o **textFieldShouldReturn:**. Tenha muito cuidado para que não haja nenhum erro de digitação ou de uso de maiúsculas e minúsculas; caso contrário, o método não será chamado. O seletor da mensagem que o campo de texto envia deve ser exatamente igual ao seletor do método implementado.

```

- (BOOL)textFieldShouldReturn:(UITextField *)textField
{
    NSLog(@"%@", textField.text);
    return YES;
}

```

Compile e execute o aplicativo, digite algum texto no campo de texto e toque na tecla de retorno. O texto deve ser exibido no console.

Observe que não foi necessário implementar *todos* os métodos delegate do campo de texto, apenas aquele que for mais importante. No tempo de execução, o campo de texto perguntará a seu delegate se ele implementou algum método antes de chamá-lo.

## Protocolos

Para cada objeto que pode ter um delegate, há um *protocolo* correspondente que declara as mensagens que o objeto pode enviar ao delegate. O delegate implementa métodos do protocolo para eventos em que está interessado. Quando uma classe implementa métodos de um protocolo, diz-se que está *em conformidade com o protocolo*.

(Se você vem do Java ou C#, você utilizaria a palavra “interface” em vez de “protocolo”.)

O protocolo para o delegate da **UITextField** é semelhante a:

```

@protocol UITextFieldDelegate <NSObject>

@optional

- (BOOL)textFieldShouldBeginEditing:(UITextField *)textField;
- (void)textFieldDidBeginEditing:(UITextField *)textField;
- (BOOL)textFieldShouldEndEditing:(UITextField *)textField;
- (void)textFieldDidEndEditing:(UITextField *)textField;
- (BOOL)textField:(UITextField *)textField
    shouldChangeCharactersInRange:(NSRange)range
        replacementString:(NSString *)string;
- (BOOL)textFieldShouldClear:(UITextField *)textField;
- (BOOL)textFieldShouldReturn:(UITextField *)textField;

@end

```

Esse protocolo, como todos os outros, é declarado com a diretiva `@protocol` seguido por seu nome, `UITextFieldDelegate`. O `NSObject` entre os colchetes angulares (`<>`) refere-se ao protocolo `NSObject` e nos diz que `UITextFieldDelegate` inclui todos os métodos no protocolo `NSObject`. Os métodos específicos para `UITextFieldDelegate` são declarados em seguida, e o protocolo termina com uma diretiva `@end`.

Observe que um protocolo não é uma classe; trata-se simplesmente de uma lista de declarações de métodos. Você não pode criar instâncias de um protocolo, ele não pode ter variáveis de instâncias, e estes métodos não são implementados em nenhum lugar no protocolo. Em vez disso, a implementação é deixada para cada classe em conformidade com o protocolo.

O protocolo `UITextFieldDelegate` faz parte do SDK do iOS. Os protocolos no SDK do iOS têm páginas de referência na documentação do desenvolvedor, nas quais você pode ver quais métodos são declarados. Você pode também escrever seu próprio protocolo. Você fará isso no Chapter 22.

Os métodos declarados em um protocolo podem ser obrigatórios ou opcionais. Por padrão, métodos de protocolo são obrigatórios. Se um protocolo tiver métodos opcionais, tais métodos serão precedidos pela diretiva `@optional`. Retornando ao protocolo `UITextFieldDelegate`, você pode perceber que todos os métodos são opcionais. Isso é geralmente verdade para protocolos de delegate.

Antes de enviar uma mensagem opcional, o objeto primeiro pergunta ao delegate se pode enviar tal mensagem, enviando outra mensagem, `respondsToSelector:`. Todo objeto implementa esse método, o qual verifica no tempo de execução se um objeto implementa determinado método. É possível transformar o seletor de métodos em um valor que você pode passar como argumento com a diretiva `@selector()`. Por exemplo, `UITextField` poderia implementar um método parecido com este:

```
- (void)clearButtonTapped
{
    // textFieldShouldClear: is an optional method,
    // so we check first
    SEL clearSelector = @selector(textFieldShouldClear);

    if ([self.delegate respondsToSelector:clearSelector]) {
        if ([self.delegate textFieldShouldClear:self]) {
            self.text = @"";
        }
    }
}
```

Se um método, em um protocolo, for obrigatório, então a mensagem será enviada sem verificar primeiro. Isso significa que se o delegate não implementar este método, será gerada uma exceção de seletor não reconhecido e o aplicativo falhará.

Para impedir que isso aconteça, o compilador insistirá que uma classe implemente os métodos obrigatórios em um protocolo. Mas, para que o compilador saiba verificar as implementações de métodos obrigatórios de um protocolo, a classe deve declarar explicitamente que está em conformidade com um protocolo. Isso é feito tanto no arquivo de cabeçalho da classe quanto na extensão da classe: os protocolos com os quais uma classe está em conformidade são adicionados a uma lista separada por vírgulas, entre colchetes angulares, na declaração de interface.

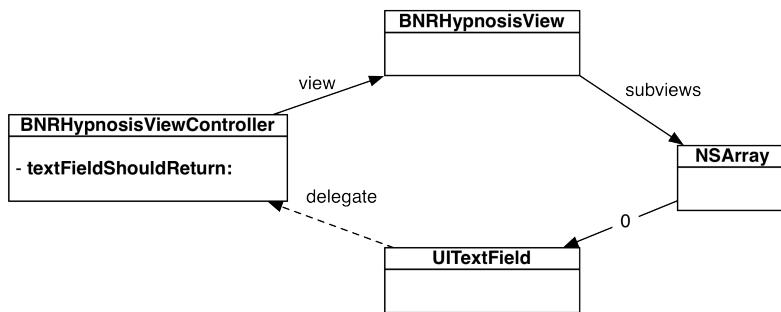
No `BNRHypnosisViewController.m`, declare que `BNRHypnosisViewController` está em conformidade com o protocolo `UITextFieldDelegate` na extensão da classe. A razão de adicioná-lo à extensão da classe em vez de ao arquivo de cabeçalho é a mesma de sempre: adicione-o à extensão de classe se as informações (em conformidade com um protocolo específico, neste caso) não precisarem estar publicamente visíveis, e adicione-o ao arquivo de cabeçalho se outros objetos não precisarem saber sobre as informações.

```
@interface BNRHypnosisViewController () <UITextFieldDelegate>
@end
```

Compile o aplicativo novamente. Agora que você declarou que `BNRHypnosisViewController` está em conformidade com o protocolo `UITextFieldDelegate`, o aviso da linha de código onde você define o delegate desaparece. Além disso, caso queira implementar métodos adicionais do protocolo `UITextFieldDelegate` em `BNRHypnosisViewController`, agora tais métodos serão completados automaticamente pelo Xcode.

Muitas classes têm um ponteiro `delegate` e quase sempre são uma referência fraca para impedir ciclos de referências fortes. Nesse caso, por exemplo, seu controlador de visão possui o campo de texto indiretamente. Se o campo de texto tivesse seu delegate, você teria um ciclo de referência forte, que causaria vazamento de memória.

Figure 7.5 Impedindo ciclo de referência forte



## Adição de rótulos à tela

Para tornar as coisas um pouco mais interessantes, você adicionará instâncias de `UILabel` à tela em posições aleatórias. No `BNRHypnosisViewController.m`, implemente um novo método que desenhará determinada string na tela vinte vezes, em posições aleatórias.

```

- (void)drawHypnoticMessage:(NSString *)message
{
    for (int i = 0; i < 20; i++) {

        UILabel *messageLabel = [[UILabel alloc] init];

        // Configure the label's colors and text
        messageLabel.backgroundColor = [UIColor clearColor];
        messageLabel.textColor = [UIColor whiteColor];
        messageLabel.text = message;

        // This method resizes the label, which will be relative
        // to the text that it is displaying
        [messageLabel sizeToFit];

        // Get a random x value that fits within the hypnosis view's width
        int width =
            (int)(self.view.bounds.size.width - messageLabel.bounds.size.width);
        int x = arc4random() % width;

        // Get a random y value that fits within the hypnosis view's height
        int height =
            (int)(self.view.bounds.size.height - messageLabel.bounds.size.height);
        int y = arc4random() % height;

        // Update the label's frame
        CGRect frame = messageLabel.frame;
        frame.origin = CGPointMake(x, y);
        messageLabel.frame = frame;

        // Add the label to the hierarchy
        [self.view addSubview:messageLabel];
    }
}
  
```

No `BNRHypnosisViewController.m`, atualize o método `textFieldShouldReturn:` para que chame esse novo método, passando o texto do campo de texto, apague o texto que o usuário digitou e dispense o teclado chamando `resignFirstResponder`.

```

- (BOOL)textFieldShouldReturn:(UITextField *)textField
{
    NSLog(@"%@", textField.text);
    [self drawHypnoticMessage:textField.text];

    textField.text = @"";
    [textField resignFirstResponder];

    return YES;
}
  
```

Compile e execute o aplicativo, e insira algum texto no campo de texto. Após tocar na tecla de retorno, o texto deverá ser exibido nas instâncias de **UILabel** na visão.

## Efeitos de movimentação

Os dispositivos iOS têm inúmeros componentes poderosos embutidos neles. Alguns deles, como o acelerômetro, o magnetômetro e o giroscópio, ajudam a determinar a orientação do dispositivo. Eles são a maneira como o dispositivo sabe, por exemplo, se a orientação da tela será retrato ou paisagem. Primeiramente no iOS 7, a Apple lançou um meio pelo qual os aplicativos podem desfrutar facilmente dos benefícios desses sensores adicionando um parallax integrado.

Quando você está dirigindo na estrada, as placas ao lado parecem se mover muito mais rápido do que as árvores mais distantes. Seu cérebro interpreta essa diferença na velocidade aparente como um movimento no espaço. O efeito visual é chamado de “parallax”. No iOS 7, provavelmente você notou esse efeito na tela inicial, em que os ícones parecem se mover em relação ao papel de parede quando você gira o dispositivo. Ele é usado de forma sutil (e não com muita sutileza) em vários locais do sistema operacional e de pacotes de aplicativos, inclusive nos emblemas vermelhos nos ícones da tela inicial, no pop-up de mudança de volume e nas visões de alerta.

Os aplicativos podem ter acesso à mesma tecnologia que gera esses efeitos com o uso da classe **UIInterpolatingMotionEffect**. São concedidos às instâncias um eixo (tanto horizontal como vertical), um percurso de chaves (que propriedade da visão você deseja impactar) e um valor relativo mínimo e máximo (até que ponto o percurso de chaves pode fazer um deslocamento para qualquer uma das direções).

No `BNRHypnosisViewController.m`, modifique o método **drawHypnoticMessage**: para adicionar um efeito de movimentação vertical e horizontal em cada rótulo que permite que seu centro se desloque 25 pontos em ambas as direções.

```
[self.view addSubview:messageLabel];

UIInterpolatingMotionEffect *motionEffect;
motionEffect = [[UIInterpolatingMotionEffect alloc] initWithKeyPath:@"center.x"
    type:UIInterpolatingMotionEffectTypeTiltAlongHorizontalAxis];
motionEffect.minimumRelativeValue = @(-25);
motionEffect.maximumRelativeValue = @(25);
[messageLabel addMotionEffect:motionEffect];

motionEffect = [[UIInterpolatingMotionEffect alloc] initWithKeyPath:@"center.y"
    type:UIInterpolatingMotionEffectTypeTiltAlongVerticalAxis];
motionEffect.minimumRelativeValue = @(-25);
motionEffect.maximumRelativeValue = @(25);
[messageLabel addMotionEffect:motionEffect];
}
```

Para testar os efeitos de movimentação, o aplicativo deve estar sendo executado em um dispositivo. Se você possui um dispositivo provisionado para uso por desenvolvedores, compile e execute o aplicativo no dispositivo. Adicione algumas mensagens hipnóticas à visão e gire levemente o dispositivo em relação ao seu rosto. Você perceberá a ilusão de óptica de profundidade que os efeitos de movimentação oferecem.

## Utilização do depurador

Quando um aplicativo é iniciado a partir do Xcode, o depurador é anexado a ele. O depurador monitora o estado atual do aplicativo, como qual método ele está executando no momento e os valores das variáveis que podem ser acessadas a partir daquele método. Usar o depurador pode ajudar a entender o que um aplicativo está fazendo realmente, que, por sua vez, ajuda a encontrar e corrigir bugs.

## Utilização de pontos de interrupção

Uma maneira de usar o depurador é definir um *ponto de interrupção*. A definição de um ponto de interrupção na linha de código pausa a execução do aplicativo naquela linha (antes da execução). Assim, é possível executar o código subsequente linha por linha. Isso é útil quando o aplicativo não está fazendo o esperado e você precisa isolar o problema.

No navegador de projetos, selecione `BNRHypnosisView.m` (não `BNRHypnosisViewController.m`). Encontre a linha de código no **initWithFrame**: que define a propriedade `circleColor` como cinza claro. Estabeleça um

ponto de interrupção, clicando na medianiz (a barra levemente sombreada no lado esquerdo da área do editor) ao lado daquela linha de código (Figure 7.6). O indicador azul mostra onde o aplicativo será “interrompido” na próxima vez que ele for executado.

Figure 7.6 Ponto de interrupção

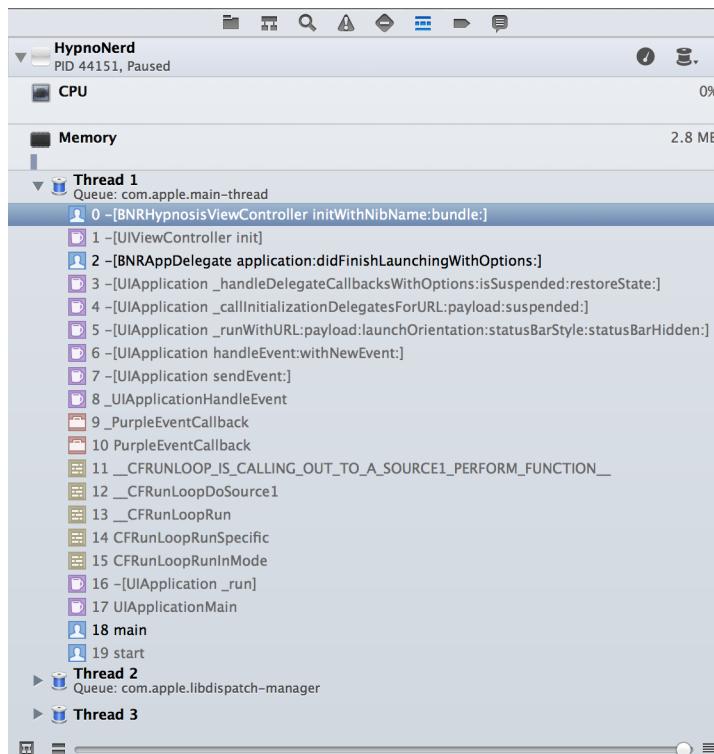
```

29
30
31
32
33
34
35 - (id)initWithNibName:(NSString *)NibNameOrNil bundle:(NSBundle *)nibBundleOrNilOrNil
36 {
37     self = [super initWithNibName:nil bundle:nil];
38
39     if (self) {
40         self.tabBarItem.title = @"Hypnosis";
41         self.tabBarItem.image = [UIImage imageNamed:@"Hypno"];
42     }
43
44     return self;
45 }
```

Compile e execute o aplicativo. O aplicativo será iniciado e então interrompido antes da execução da linha de código onde o ponto de interrupção foi inserido. Observe o indicador verde claro e o sombreado que aparece para mostrar o ponto de execução atual.

Agora, seu aplicativo está temporariamente congelado no tempo, sendo possível examiná-lo mais de perto. Na área de navegação, clique na guia para abrir o *navegador de depuração*. Esse navegador mostra um *rastreamento de pilha* que começa onde o ponto de interrupção parou a execução (Figure 7.7). O rastreamento de pilha mostra os métodos e funções cujos frames estavam na pilha quando o aplicativo foi interrompido. O controle deslizante na parte inferior do navegador de depuração expande e retrai a pilha. Arraste-o para a direita para ver todos os métodos no rastreamento de pilha.

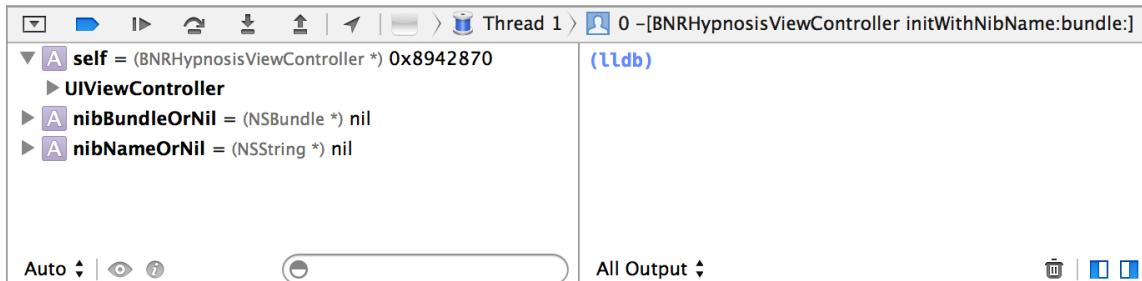
Figure 7.7 O navegador de depuração



O método no qual a interrupção ocorreu está na parte superior do rastreamento de pilha. Ele foi chamado pelo método imediatamente abaixo dele, que foi chamado pelo método imediatamente abaixo dele, e assim por diante. Observe que os métodos para os quais você escreveu códigos estão em texto preto, e os métodos que pertencem à Apple estão em cinza.

Selecione o método na parte superior da pilha. Na área de depuração, abaixo da área do editor, verifique a visão de variáveis à esquerda do console. Essa área mostra as variáveis no escopo do método `initWithFrame:` de `BNRHypnosisView`, juntamente com seus valores atuais (Figure 7.8).

Figure 7.8 Área de depuração com visão de variáveis



(Se a visão de variáveis não estiver sendo exibida, localize o controle no canto inferior direito do console. Clique no ícone à esquerda para exibir a visão de variáveis.)

Na visão de variáveis, uma variável que é um ponteiro mostra o endereço do objeto. Você pode ver que `self` possui endereço porque no contexto desse método, `self` é um ponteiro para a instância de `BNRHypnosisView`, e essa instância foi alocada antes da interrupção do aplicativo.

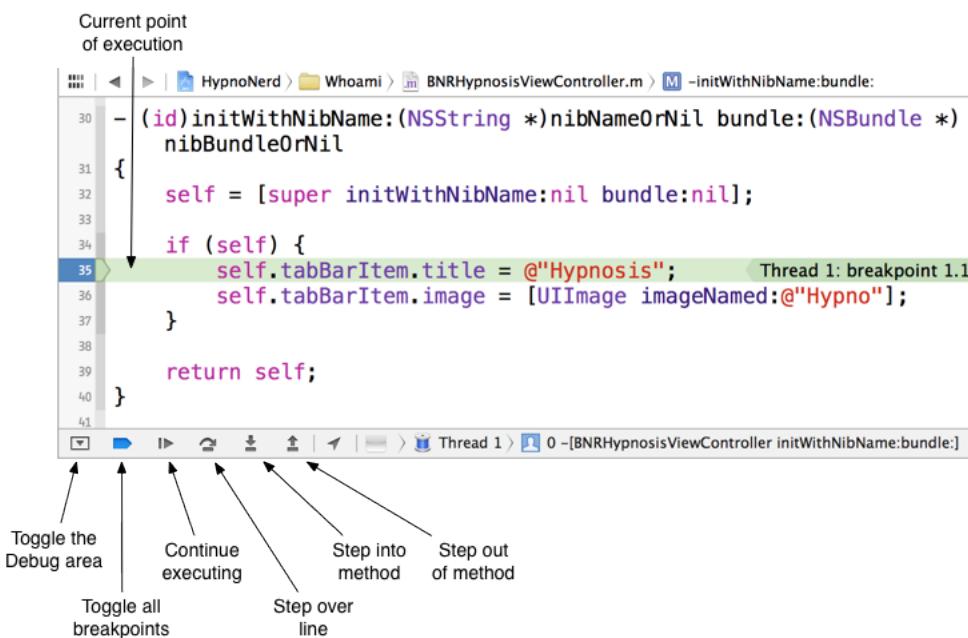
Clique no botão de divulgação ao lado de `self`. O primeiro item abaixo de `self` é a superclasse. A superclasse de `BNRHypnosisView` é `UIViewController`. Clicar no botão de divulgação ao lado de `UIViewController` exibirá as variáveis que `self` herda de sua superclasse.

`BNRHypnosisView` possui uma variável própria: `_circleColor`. O ponto de interrupção é definido na linha em que a propriedade `circleColor` é definida. Essa linha de código ainda precisa ser executada, portanto, `_circleColor` atualmente aponta para `nil`.

## Percorrendo o código

Além de fornecer um instantâneo do aplicativo em determinado momento, o depurador também permite que você *percorra* seu código, linha a linha, e veja o que seu aplicativo faz conforme cada linha é executada. Os botões que controlam a execução se encontram na *barra do depurador*, que fica entre a área do editor e a área de depuração (Figure 7.9).

Figure 7.9 Barra do depurador



Clique no botão que percorre cada linha. Isso executará apenas a linha atual do código, que define `circleColor`. Observe que o indicador de execução verde e o sombreamento passam para a próxima linha. E ainda mais interessante, a visão de variáveis mostra que o valor de `_circleColor` foi alterado para um endereço válido.

Nesse ponto, você poderia continuar percorrendo o código para ver o que acontece. Ou poderia clicar no botão para continuar a executar o código normalmente. Ou poderia percorrer um método. Percorrer um método leva você ao método que é chamado pela linha do código que está atualmente com o indicador de execução verde. Quando está no método, você tem a oportunidade de percorrer seu código da mesma maneira.

Quando você sai de um método, você é direcionado ao método que o chamou. Para experimentar, clique no botão para sair do método atual. Você será direcionado à implementação de `loadView` no `BNRHypnosisViewController.m`.

## Delegação de pontos de interrupção

Para executar o aplicativo normalmente de novo, você terá de se livrar do ponto de interrupção. Clique com o botão direito no indicador azul e selecione Delete Breakpoint. É possível compilar e executar para confirmar se o aplicativo está sendo executado como o esperado.

Às vezes, um desenvolvedor estabelece um ponto de interrupção e se esquece dele. Então, quando o aplicativo é executado, a execução é interrompida e parece que o aplicativo falhou. Se um de seus aplicativos parar inesperadamente, certifique-se de que não foi devido a um ponto de interrupção esquecido.

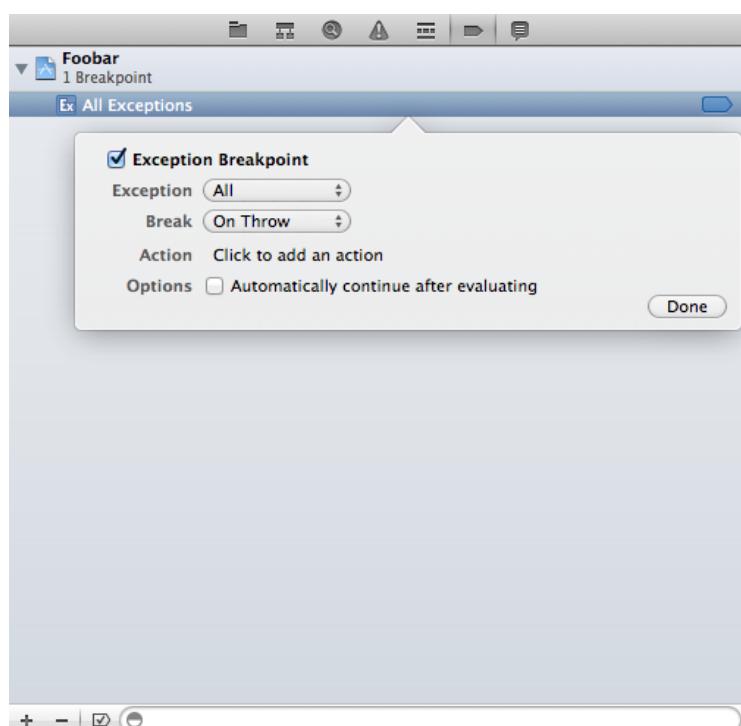
Se você não sabe ao certo onde possa ter deixado o ponto de interrupção, poderá ver uma lista de pontos de interrupção de seu projeto no navegador de pontos de interrupção (a guia ➔ na área do navegador).

## Definição de um ponto de interrupção de exceção

Você também pode dizer ao depurador para que faça a interrupção automaticamente em qualquer linha que faça seu aplicativo travar ou causar uma exceção.

Na área de navegação, selecione a guia ➔ para abrir o navegador de pontos de interrupção. Na parte inferior desse navegador, clique no ícone + e selecione Add Exception Breakpoint....

Figure 7.10 Adição de um ponto de interrupção de exceção



Se seu aplicativo estiver disparando exceções e você não souber o motivo, adicionar um ponto de interrupção de exceção ajudará você a descobrir o que está acontecendo.

## Para os mais curiosos: main() e UIApplication

Um aplicativo em C começa executando uma função `main`. Os aplicativos em Objective-C não são diferentes, mas você ainda não viu `main()` em nenhum dos seus aplicativos para iOS. Vamos dar uma olhada agora.

Abra o `main.m` no navegador de projetos do HypnoNerd. Isso será semelhante ao seguinte:

```
int main(int argc, char *argv[])
{
    @autoreleasepool {
        return UIApplicationMain(argc, argv,
                               nil, NSStringFromClass([BNRAppDelegate class]));
    }
}
```

A função `UIApplicationMain` cria uma instância de uma classe chamada `UIApplication`. Para cada aplicativo, há uma única instância de `UIApplication`. Esse objeto é responsável pela manutenção do loop de execução. Quando o objeto do aplicativo é criado, seu loop de execução basicamente se torna um loop infinito: o segmento em execução nunca retorna para `main()`.

Outra coisa que a função `UIApplicationMain` faz é criar uma instância da classe que servirá de delegate da `UIApplication`. Observe que o argumento final da função `UIApplicationMain` é uma `NSString` que é o nome da classe do delegate. Portanto, essa função cria uma instância de `BNRAppDelegate` e a define como a delegate do objeto `UIApplication`.

O primeiro evento adicionado ao loop de execução de cada aplicativo é um evento de “lançamento” especial que faz o aplicativo enviar uma mensagem para seu delegate. Essa mensagem é o `application:didFinishLaunchingWithOptions:`. Você implementou esse método no `BNRAppDelegate.m` para criar a janela e os objetos controladores usados neste aplicativo.

Todos os aplicativos de iOS seguem esse padrão. Se você ainda está curioso, volte e verifique o arquivo `main.m` no aplicativo Quiz que você escreveu no Chapter 1.

## Desafio de prata: pinça para zoom

Adicione a pinça para zoom no projeto do Hypnosister do Chapter 5.

A primeira etapa é conceder um delegate à visão com rolagem:

- `BNRAppDelegate` deve estar em conformidade com o protocolo `UIScrollViewDelegate`.
- No `application:didFinishLaunchingWithOptions:`, defina a propriedade de delegate da visão com rolagem.

Para trabalhar como o delegate da visão com rolagem, a `BNRAppDelegate` precisará de uma propriedade que aponte para a instância de `BNRHypnosisView`. Adicione essa propriedade em uma extensão de classe no `BNRAppDelegate.m` e atualize o restante do código para usar a propriedade em vez da variável local `BNRHypnosisView`.

Para configurar a visão com rolagem, você precisará conceder a ela uma `BNRHypnosisView` como subvisão e desativar a paginação. A visão com rolagem também precisa de limites para a quantidade de zoom de aproximação e afastamento. Localize as propriedades `UIScrollView` relevantes para definir a página de referência dessa classe na documentação.

Finalmente, você precisará implementar o método delegate `viewForZoomingInScrollView:` da visão com rolagem para retornar a `BNRHypnosisView`.

Se você ficar preso, visite as páginas de referência para a classe `UIScrollView` e para o protocolo `UIScrollViewDelegate`.

Para simular dois dedos e testar o zoom, no simulador, mantenha pressionada a tecla Option enquanto usa o mouse.

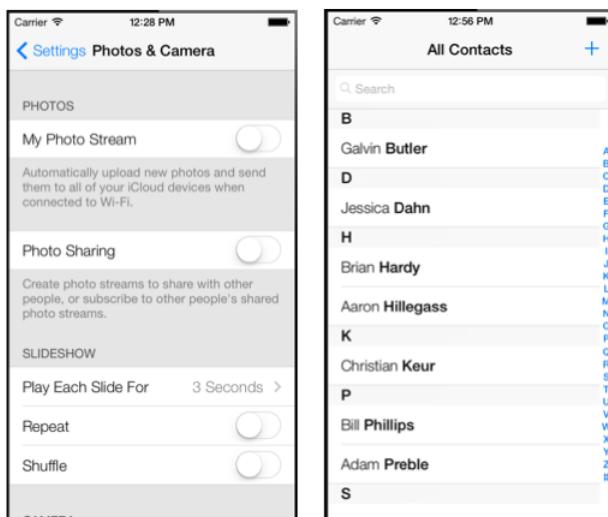
# 8

## UITableView e UITableViewController

Muitos aplicativos de iOS mostram ao usuário uma lista de itens e permitem que ele selecione, exclua ou reordene os itens da lista. Seja em um aplicativo que exibe uma lista de pessoas nos contatos do usuário ou uma lista de itens na App Store, é a **UITableView** que faz esse trabalho.

Uma **UITableView** exibe uma única coluna de dados com um número variável de linhas. A Figure 8.1 mostra alguns exemplos da **UITableView**.

Figure 8.1 Exemplos da **UITableView**

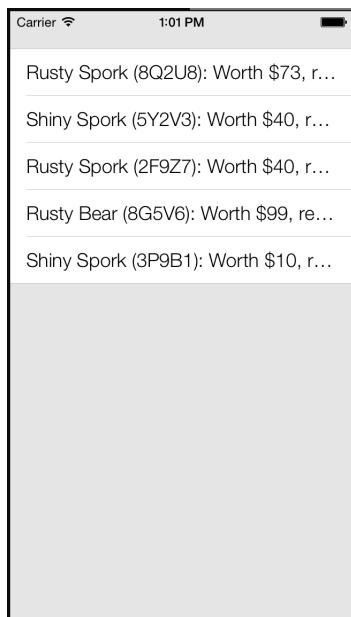


### Iniciando o aplicativo Homepwner

Neste capítulo, você vai começar um aplicativo chamado Homepwner, que mantém um inventário de tudo o que você possui. No caso de um incêndio ou outra catástrofe, você terá um registro para sua seguradora. (A propósito, o nome “Homepwner” não tem nenhum erro de digitação. Se você quiser uma definição do termo “pwn”, consulte [www.urbandictionary.com](http://www.urbandictionary.com).)

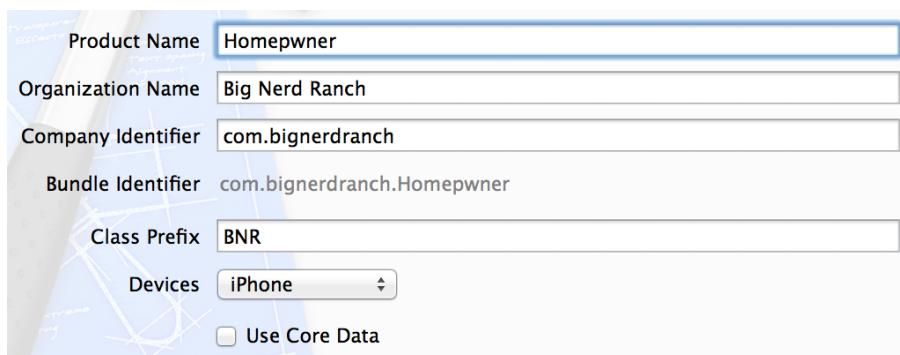
Até agora, seus projetos de iOS foram pequenos, mas o Homepwner se tornará um aplicativo realisticamente complexo ao longo de nove capítulos. No fim deste capítulo, o Homepwner apresentará uma lista de objetos **BNRItem** em uma **UITableView**, conforme mostrado na Figure 8.2.

Figure 8.2 Homepwner: fase 1



Crie um novo projeto iOS Empty Application e configure-o conforme mostrado na Figure 8.3.

Figure 8.3 Configuração do Homepwner



## UITableViewController

Uma **UITableView** é um objeto de visão. Lembre-se de que no padrão de projeto Modelo-Visão-Controlador, que os desenvolvedores de iOS fazem seu melhor para seguir, cada classe é exatamente um dos seguintes:

- *Modelo*: armazena dados e não sabe nada sobre a interface do usuário.
- *Visão*: é visível para o usuário e não sabe nada sobre os objetos de modelo.
- *Controlador*: mantém a interface do usuário e os objetos de modelo sincronizados. Controla o fluxo do aplicativo; por exemplo, o controlador pode ser responsável por mostrar uma mensagem “Tem certeza de que quer excluir esse item?” antes de realmente excluir algum dado.

Assim, uma **UITableView**, que é um objeto de visão, não lida com lógica ou dados do aplicativo. Ao usar uma **UITableView**, você deve considerar o que mais é necessário para fazer com que a tabela funcione no seu aplicativo.

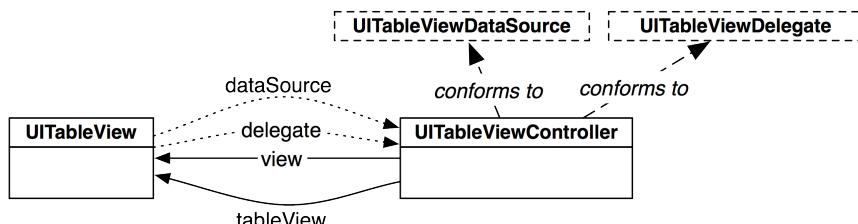
- Uma **UITableView** geralmente precisa de um controlador de visão para administrar sua aparência na tela.
- Uma **UITableView** precisa de uma *fonte de dados*. Uma **UITableView** solicita à sua fonte de dados algumas linhas para exibir, os dados a serem mostrados nessas linhas e outras coisinhas que tornam a **UITableView** uma interface de usuário funcional. Sem a fonte de dados, a visão de tabela é apenas um recipiente vazio. A **dataSource** para uma **UITableView** pode ser qualquer tipo de objeto do Objective-C, contanto que esteja em conformidade com o protocolo **UITableViewDataSource**.
- Uma **UITableView** geralmente precisa de um *delegate* que possa informar outros objetos dos eventos que envolvem a **UITableView**. O delegate pode ser qualquer objeto, contanto que esteja em conformidade com o protocolo **UITableViewDelegate**.

Uma instância da classe **UIViewController** pode exercer as três funções: controlador de visão, fonte de dados e delegate.

A **UIViewController** é uma subclasse de **UIViewController** e, portanto, a **UIViewController** tem uma view. A view de uma **UIViewController** é sempre uma instância de **UITableView**, e a **UIViewController** cuida da preparação e apresentação da **UITableView**.

Quando uma **UIViewController** cria sua visão, as variáveis de instância **dataSource** e **delegate** da **UITableView** são automaticamente definidas para apontar para a **UIViewController** (Figure 8.4).

Figure 8.4 Relacionamento **UIViewController**-**UITableView**



## Criação de subclasses de UITableViewcontroller

Agora, você vai escrever uma subclasse de **UIViewController** para o Homepwner. Para esse controlador de visão, usaremos o template **NSObject**. No menu File, selecione New e depois File.... Na seção iOS, selecione Cocoa Touch, escolha Objective-C class e clique em Next. Depois, selecione NSObject no menu pop-up e digite **BNRItemsViewController** como o nome da nova classe. Clique em Next e depois clique em Create na próxima folha para salvar sua classe.

Abra o `BNRItemsViewController.h` e altere sua superclasse:

```
#import <Foundation/Foundation.h>
@interface BNRItemsViewController : NSObject

#import <UIKit/UIKit.h>

@interface BNRItemsViewController : UITableViewController
```

O inicializador designado de `UITableViewController` é o `initWithStyle:`, que aceita uma constante que determina o estilo da visão de tabela. Há duas opções: `UITableViewStylePlain` e `UITableViewStyleGrouped`. Elas eram bem diferentes no iOS 6, mas agora, no iOS 7, as diferenças são bem pequenas.

Você está mudando o inicializador designado para `init`. Sendo assim, você deve seguir as duas regras de inicializadores:

- Chamar o inicializador designado da superclasse a partir do seu
- Sobrescrever o inicializador designado da superclasse para chamar seu

Faça as duas coisas no `BNRItemsViewController.m`:

```
#import "BNRItemsViewController.h"

@implementation BNRItemsViewController

- (instancetype)init
{
    // Call the superclass's designated initializer
    self = [super initWithStyle:UITableViewStylePlain];
    return self;
}

- (instancetype)initWithStyle:(UITableViewStyle)style
{
    return [self init];
}
```

Isso assegurará que todas as instâncias de `BNRItemsViewController` utilizem o estilo `UITableViewStylePlain`, independentemente da mensagem de inicialização enviada a elas.

Abra o `BNRAppDelegate.m`. No `application:didFinishLaunchingWithOptions:`, crie uma instância de `BNRItemsViewController` e defina essa instância como a `rootViewController` da janela. Certifique-se de importar o arquivo de cabeçalho da `BNRItemsViewController` no topo desse arquivo.

```
#import "BNRItemsViewController.h"

@implementation BNRAppDelegate

- (BOOL)application:(UIApplication *)application
    didFinishLaunchingWithOptions:(NSDictionary *)launchOptions
{
    self.window = [[UIWindow alloc] initWithFrame:[[UIScreen mainScreen] bounds]];
    // Override point for customization after application launch

    // Create a BNRItemsViewController
    BNRItemsViewController *itemsViewController =
        [[BNRItemsViewController alloc] init];

    // Place BNRItemsViewController's table view in the window hierarchy
    self.window.rootViewController = itemsViewController;

    self.window.backgroundColor = [UIColor whiteColor];
    [self.window makeKeyAndVisible];
    return YES;
}
```

Compile e execute seu aplicativo. Você deverá ver uma tela vazia, como mostrado na Figure 8.5; mas há uma visão de tabela vazia lá. Como subclasse da **UIViewController**, uma **UITableViewController** herda o método **view**. Esse método chama o **loadView**, que cria e carrega um objeto de visão vazio caso não exista nenhum. A view de uma **UITableViewController** é sempre uma instância de **UITableView**; portanto, enviar **view** à **UITableViewController** leva você a uma visão de tabela clara, brilhante e vazia.

Figure 8.5 **UITableView** vazia



Sua visão de tabela precisa de algumas linhas para serem exibidas. Você se lembra da classe **BNRItem** que escreveu no Chapter 2? Agora, você vai usar novamente essa classe: cada linha da visão de tabela exibe uma instância de **BNRItem**.

Localize os arquivos de cabeçalho e de implementação da **BNRItem** (**BNRItem.h** e **BNRItem.m**) no Finder e arraste-os para o navegador de projetos do Homepwner.

Ao arrastar esses arquivos para sua janela de projeto, selecione a caixa identificada como **Copy items into destination group's folder** quando solicitado. Isso copia os arquivos de seu diretório atual para o diretório do seu projeto no sistema de arquivos e os adiciona ao seu projeto.

Você não irá mais precisar das propriedades **container** ou **containedItem** (elas estavam lá apenas para demonstrar ciclos de referências fortes), então exclua-as do **BNRItem.h**:

```
@property (nonatomic, strong) BNRItem *containedItem;
@property (nonatomic, weak) BNRItem *container;
```

Exclua também o método **setContainedItem:** no **BNRItem.m**:

```
- (void)setContainedItem:(BNRItem *)i
{
    _containedItem = i;
    // When given an item to contain, the contained
    // item will be given a pointer to its container
    self.containedItem.container = self;
}
```

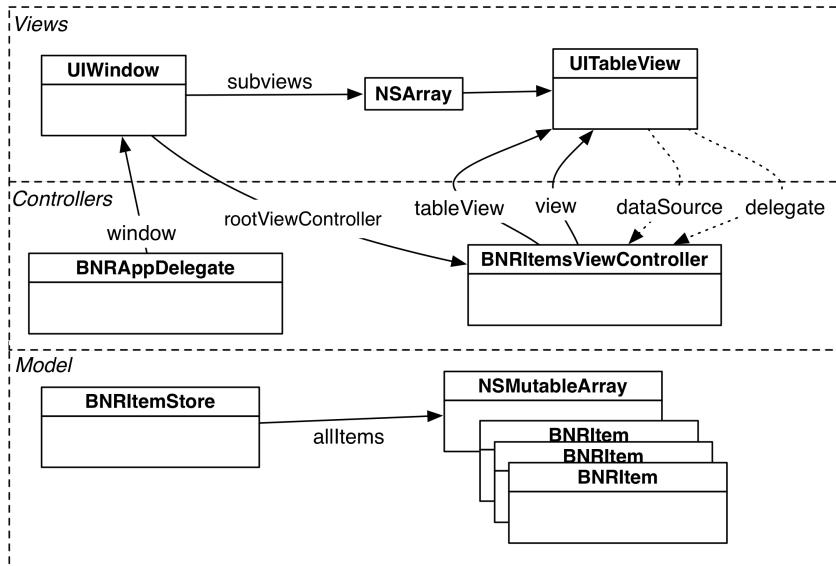
## Fonte de dados da UITableView

O processo de criar linhas na **UITableView** no Cocoa Touch é diferente da tarefa típica de programação baseada em procedimentos. Em um design baseado em procedimentos, você diz à visão de tabela o que ela deve exibir. No Cocoa Touch, a visão de tabela pergunta a outro objeto (sua **dataSource**) o que deve exibir. No nosso

caso, a **BNRItemsViewController** é a fonte de dados, então ela precisa de uma maneira de armazenar dados de itens.

No Chapter 2, você usou uma **NSMutableArray** para armazenar instâncias de **BNRItem**. Você vai fazer a mesma coisa neste capítulo, mas com uma pequena diferença. A **NSMutableArray** que armazena as instâncias de **BNRItem** será abstraída em outro objeto: uma **BNRItemStore** (Figure 8.6). Por que não usar simplesmente um array? Eventualmente, o objeto **BNRStore** também cuidará do salvamento e carregamento dos itens.

Figure 8.6 Diagrama de objetos do Homepwner



Se um objeto quer ver todos os itens, ele pede à **BNRItemStore** o array que os contém. Em capítulos posteriores, você tornará o armazenamento responsável pela realização de operações no array, como reordenação, adição e remoção de itens. Ele também será responsável pelo salvamento e carregamento de itens do disco.

## Criação de uma **BNRItemStore**

No menu File, selecione New e depois File.... Crie uma nova subclasse **NSObject** e nomeie-a como **BNRItemStore**.

A **BNRItemStore** será um singleton. Isso significa que só haverá uma instância desse tipo no aplicativo; se você tentar criar outra instância, a classe silenciosamente retornará a instância que já existe. Um singleton é útil quando você tem um objeto com o qual muitos objetos vão conversar. Esses objetos podem pedir à classe singleton sua única instância.

Para obter a (única instância de) **BNRItemStore**, você enviará à classe **BNRItemStore** a mensagem **sharedStore**.

No **BNRItemStore.h**, declare **sharedStore**.

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

@interface BNRItemStore : NSObject

// Notice that this is a class method and prefixed with a + instead of a -
+ (instancetype)sharedStore;

@end
```

Quando essa mensagem é enviada à classe **BNRItemStore**, a classe verifica se a única instância de **BNRItemStore** já foi criada. Se sim, a classe retornará a instância. Se não, ela irá criar a instância e a retornar.

No **BNRItemStore.m**, implemente **sharedStore**, um método **init** que dispara uma exceção, e um inicializador designado secreto chamado **initPrivate**.

```

@implementation BNRItemStore

+ (instancetype)sharedStore
{
    static BNRItemStore *sharedStore = nil;

    // Do I need to create a sharedStore?
    if (!sharedStore) {
        sharedStore = [[self alloc] initPrivate];
    }

    return sharedStore;
}

// If a programmer calls [[BNRItemStore alloc] init], let him
// know the error of his ways
- (instancetype)init
{
    @throw [NSError exceptionWithName:@"Singleton"
                                reason:@"Use +[BNRItemStore sharedStore]"
                                userInfo:nil];
    return nil;
}

// Here is the real (secret) initializer
- (instancetype)initPrivate
{
    self = [super init];
    return self;
}

```

Observe que a variável `sharedStore` é declarada como `static` (estática). Uma variável *estática* não é destruída quando o método tiver terminado a execução. Como uma variável global, ela não é mantida na pilha.

O valor inicial da `sharedStore` é `nil`. Na primeira vez em que o método `sharedStore` é chamado, uma instância de `BNRItemStore` será criada, e a `sharedStore` será definida para apontar para ela. Em chamadas subsequentes a esse método, a `sharedStore` continuará apontando para essa instância de `BNRItemStore`. Essa variável tem uma referência forte para a `BNRItemStore` e, como essa variável nunca será destruída, o objeto para o qual ela aponta também nunca será destruído.

O controlador `BNRItemsViewController` enviará uma mensagem à `BNRItemStore` quando quiser que uma nova `BNRItem` seja criada. A `BNRItemStore` obedecerá, criará o objeto e o adicionará a um array de instâncias de `BNRItem`. A `BNRItemsViewController` também pedirá à `BNRItemStore` todos os itens do armazenamento quando quiser preencher sua `UITableView`.

No `BNRItemStore.h`, declare um método e uma propriedade com esses objetivos.

```

#import <Foundation/Foundation.h>

@class BNRItem;

@interface BNRItemStore : NSObject

@property (nonatomic, readonly) NSArray *allItems;

+ (instancetype)sharedStore;
- (BNRItem *)createItem;

@end

```

Está vendo a diretiva `@class`? Ela diz ao compilador que existe uma classe `BNRItem` e que ele não precisa conhecer os detalhes dessa classe no arquivo atual; precisa apenas saber que ela existe. Isso permite a você usar o símbolo da `BNRItem` na declaração do `createItem` sem importar o `BNRItem.h`. O uso da diretiva `@class` pode acelerar consideravelmente os tempos de compilação, porque menos arquivos precisarão ser recompilados quando um deles for alterado.

Nos arquivos que efetivamente enviam mensagens à classe `BNRItem` ou a suas instâncias, você deve importar o arquivo em que ela foi declarada para que o compilador tenha todos os detalhes dela. No topo do

`BNRItemStore.m`, importe o `BNRItem.h`, já que ele terá de enviar mensagens para as instâncias de `BNRItem` em algum momento.

```
#import "BNRItemStore.h"
#import "BNRItem.h"
```

É aqui que as coisas ficam um tanto interessantes. Você tem uma `BNRItemStore` que irá supervisionar o array de itens; isso inclui a adição de itens ao array e, mais tarde, incluirá sua remoção e reordenação. Como a `BNRItemStore` quer esse tipo de controle sobre o array, ela retorna uma `NSArray` imutável para representar o array de itens e declara a propriedade como `readonly` (somente leitura). Nenhum outro objeto pode mudar a propriedade `allItems` da `BNRItemStore`, seja dando a ela um novo array ou modificando o array que possui.

No entanto, internamente, a `BNRItemStore` precisa ter a capacidade de modificar o array para adicionar novos itens (e mais tarde removê-los e reordená-los). É um projeto bem comum para uma classe que quer um controle estrito sobre seus dados internos: um objeto se prende a uma estrutura de dados mutável, mas outros objetos só têm acesso a uma versão imutável dele.

No `BNRItemStore.m`, declare um array mutável na extensão de classe.

```
#import "BNRItem.h"

@interface BNRItemStore ()
@property (nonatomic) NSMutableArray *privateItems;
@end

@implementation BNRItemStore
```

Implemente o `initPrivate` para criar imediatamente instâncias de `privateItems`. Além disso, sobrescreva o getter de `allItems` para retornar a `privateItems`.

```
- (instancetype)initPrivate
{
    self = [super init];
    if (self) {
        _privateItems = [[NSMutableArray alloc] init];
    }
    return self;
}

- (NSArray *)allItems
{
    return self.privateItems;
}
```

Isso é possível porque a `NSMutableArray` é uma subclasse de `NSArray`. Logo, uma `NSMutableArray` é uma `NSArray`, porque pode fazer tudo o que uma `NSArray` pode fazer. (Observe que isso não funcionaria se a propriedade retornasse uma `NSMutableArray`, mas a variável de instância fosse uma `NSArray`, pois uma `NSArray` não pode fazer tudo o que sua correspondente mutável pode.)

Há um problema: mesmo que a propriedade `allItems` diga que está retornando uma `NSArray`, você sabe que todos os objetos em Objective-C conhecem o seu tipo e que o tipo de uma variável ou valor de retorno não muda esse tipo. Assim, qualquer objeto que envia `allItems` à `BNRItemStore` receberá de volta uma instância de `NSMutableArray`, mesmo que o objeto possa não saber isso.

Ao usar uma classe, tal como `BNRItemStore`, você pode confiar apenas no que o arquivo de interface lhe diz a respeito da interação com essa classe. Se o arquivo de interface lhe diz que um objeto é uma `NSArray`, você deve tratá-lo como uma `NSArray`. Agir de outro modo violaria o contrato que a `BNRItemStore` especifica com a sua interface pública. Contudo, se você estivesse sendo extremamente cuidadoso, poderia sobreescrivê-lo para retornar uma cópia imutável de sua propriedade `privateItems`. Você escreveria o código da seguinte forma (mas não faça isso aqui, pois você irá se basear em convenções em vez de regras rígidas):

```
- (NSArray *)allItems
{
    return [self.privateItems copy];
}
```

Agora que estamos livres dessa discussão, implemente o `createItem` no `BNRItemStore.m`.

```
- (BNRItem *)createItem
{
    BNRItem *item = [BNRItem randomItem];
    [self.privateItems addObject:item];
    return item;
}
```

Uma particularidade interessante desse exemplo: sequer existe uma variável de instância `_allItems`. Você declarou uma propriedade `allItems`, mas depois você implementou seus próprios acessores (bem, apenas um: a `allItems` foi declarada `readonly`). O compilador só sintetiza automaticamente uma variável de instância se você deixá-lo sintetizar pelo menos um acessor.

## Implementação de métodos de fonte de dados

No `BNRItemsViewController.m`, importe o `BNRItemStore.h` e o `BNRItem.h` e atualize o inicializador designado para adicionar 5 itens aleatórios à `BNRItemStore`.

```
#import "BNRItemsViewController.h"
#import "BNRItemStore.h"
#import "BNRItem.h"

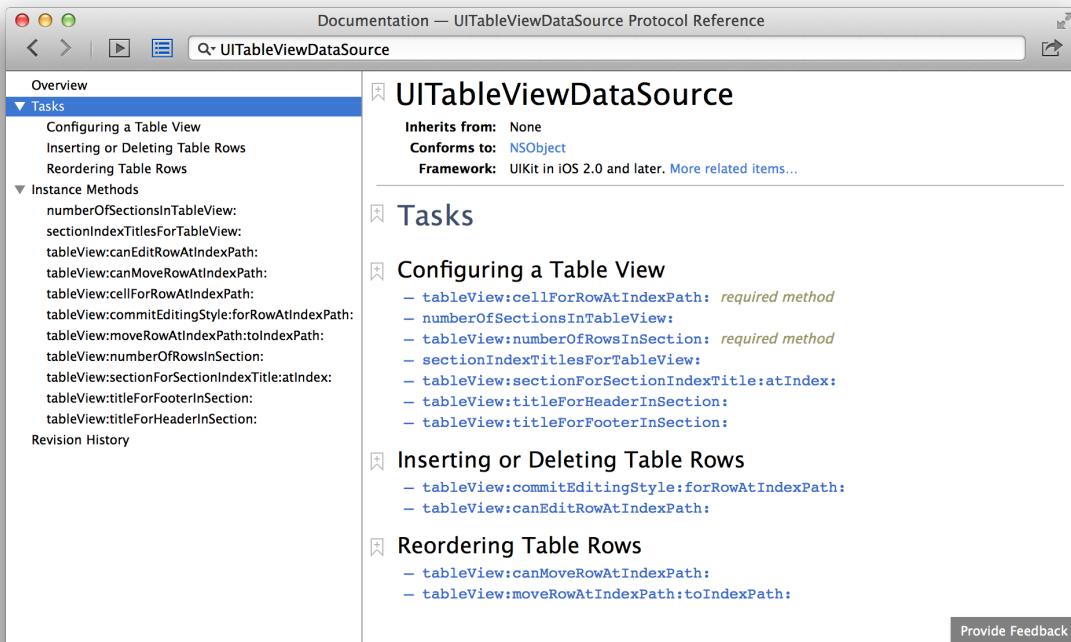
@implementation BNRItemsViewController

- (instancetype)init
{
    // Call the superclass's designated initializer
    self = [super initWithStyle:UITableViewStylePlain];
    if (self) {
        for (int i = 0; i < 5; i++) {
            [[BNRItemStore sharedStore] createItem];
        }
    }
    return self;
}
```

Agora que existem alguns itens no armazenamento, você precisa ensinar à **BNRItemsViewController** como transformar esses itens em linhas que sua **UITableView** possa exibir. Quando uma **UITableView** quer saber o que exibir, ela envia uma mensagem do conjunto de mensagens declaradas no protocolo **UITableViewDataSource**.

A partir do menu Help, selecione Documentation and API Reference. Procure a referência do protocolo **UITableViewDataSource** e então selecione Tasks no painel à esquerda (Figure 8.7).

Figure 8.7 Documentação do protocolo **UITableViewDataSource**



Na tarefa **Configuring a Table View**, observe os dois métodos marcados com *required method*. Para que a **BNRItemsViewController** esteja em conformidade com o **UITableViewDataSource**, ela deve implementar o **tableView:numberOfRowsInSection:** e o **tableView:cellForRowAtIndexPath:**. Esses métodos informam à visão de tabela quantas linhas ela deve exibir, e qual conteúdo ela deve exibir em cada linha.

Sempre que uma **UITableView** precisa ser exibida, ela envia uma série de mensagens (os métodos obrigatórios mais outros adicionais que tenham sido implementados) para sua **dataSource**. O método obrigatório **tableView:numberOfRowsInSection:** retorna um valor inteiro para o número de linhas que a **UITableView** deve exibir. Na visão de tabela do Homeowner, deve haver uma linha para cada entrada no armazenamento.

No **BNRItemsViewController.m**, implemente **tableView:numberOfRowsInSection:**.

```

- (NSInteger)tableView:(UITableView *)tableView
 numberOfRowsInSection:(NSInteger)section
{
    return [[[BNRItemStore sharedStore] allItems] count];
}

```

Observe que esse método retorna um `NSInteger`. Quando a Apple começou a suportar tanto sistemas 32 bits quanto 64 bits, precisou de um tipo de inteiro que fosse um `int` de 32 bits em aplicativos de 32 bits e um `int` de 64 bits em aplicativos de 64 bits. Assim, nasceram os tipos `NSInteger` (que é `signed`, com sinal) e `NSUInteger` (que é `unsigned`, sem sinal). Esses tipos são bastante usados nos frameworks da Apple.

Está se perguntando qual é a seção à qual esse método se refere? As visões de tabela podem ser divididas em seções, e cada seção tem seu próprio conjunto de linhas. Por exemplo, nos contatos, todos os nomes que começam com “D” são agrupados em uma seção. Por padrão, uma visão de tabela tem uma seção e, neste capítulo, você trabalhará com apenas uma. Depois que você compreende como funciona a visão de tabela, não é difícil trabalhar com várias seções. Na verdade, o uso de seções é o primeiro desafio no final deste capítulo.

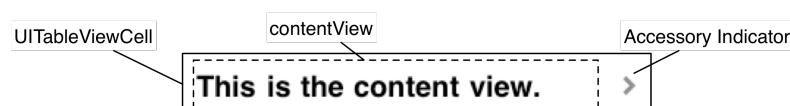
O segundo método obrigatório no protocolo `UITableViewDataSource` é o `tableView:cellForRowAtIndexPath:`. Para implementar esse método, você precisará aprender outra classe, a `UITableViewCell`.

## UITableViewCells

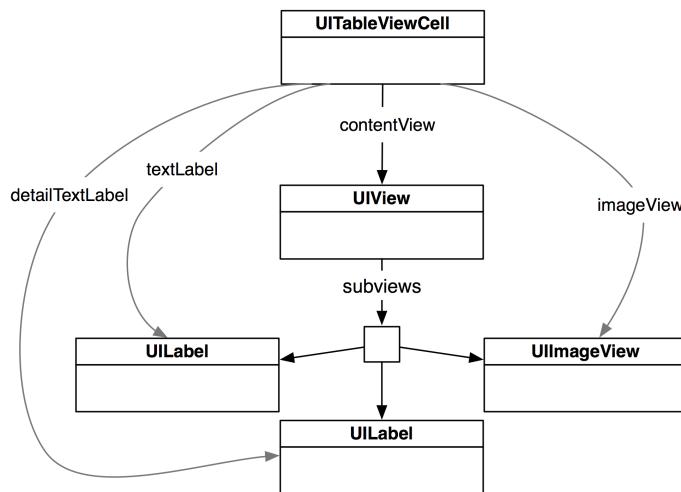
Cada linha da visão de tabela é uma visão. Essas visões são instâncias de `UITableViewCell`. Nesta seção, você criará as instâncias de `UITableViewCell` para preencher a visão de tabela. No Chapter 19, você criará uma subclasse personalizada de `UITableViewCell`.

Uma célula tem, ela mesma, uma subvisão: sua `contentView` (Figure 8.8). A `contentView` é a supervisão do conteúdo da célula. A célula também pode desenhar um indicador acessório. O indicador acessório mostra um ícone orientado a ações, como uma marca de verificação, um ícone de revelação ou um ponto azul muito bonito com um V no meio. Esses ícones são acessados por constantes predefinidas para a aparência do indicador acessório. O padrão é `UITableViewCellAccessoryNone`, e é esse que você usará neste capítulo. Mas você verá o indicador acessório novamente no Chapter 19. (Ficou curioso? Veja a página de referência sobre `UITableViewCell` para mais detalhes.)

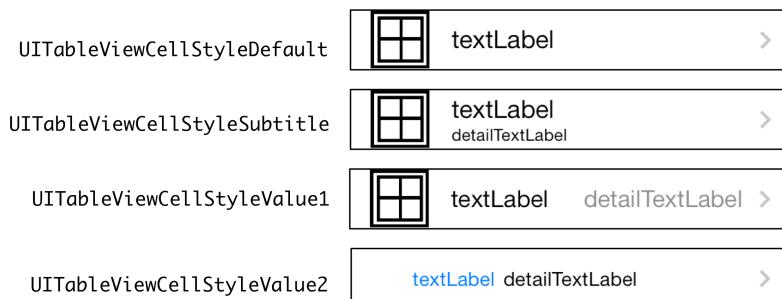
Figure 8.8 Layout da `UITableViewCell`



O verdadeiro cerne de uma `UITableViewCell` são as três subvisões da `contentView`. Duas dessas subvisões são instâncias de `UILabel` que são propriedades da `UITableViewCell` chamadas `textLabel` e `detailTextLabel`. A terceira subvisão é uma `UIImageView` chamada `imageView` (Figure 8.9). Neste capítulo, você usará apenas a `textLabel`.

Figure 8.9 Hierarquia da **UITableViewCell**

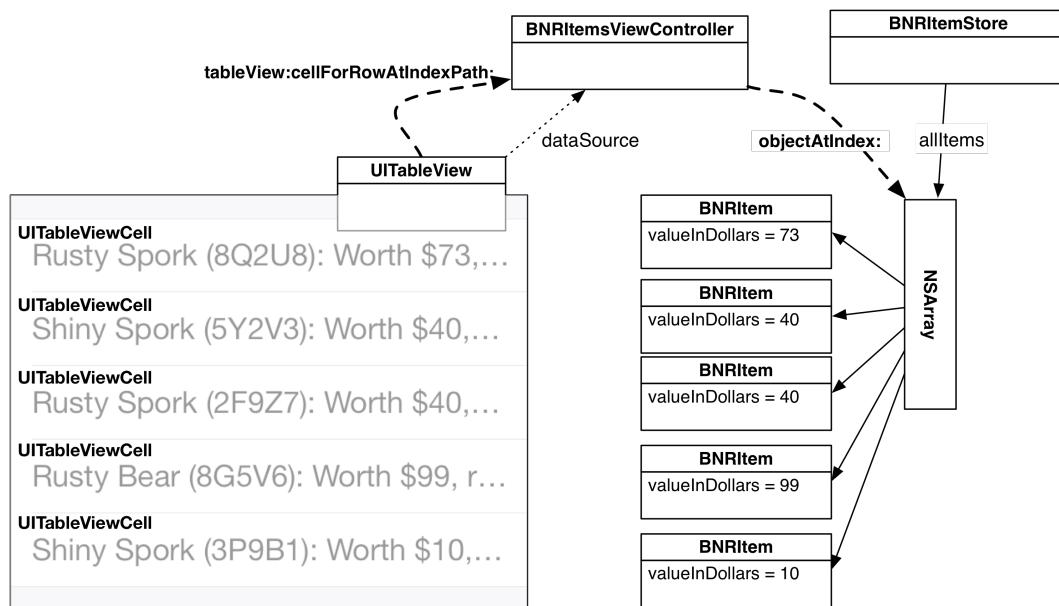
Cada célula também tem uma `UITableViewCellStyle` que determina quais subvisões são usadas e a posição delas na `contentView`. Exemplos desses estilos e das respectivas constantes são mostrados na Figure 8.10.

Figure 8.10 **UITableViewCellStyles**

## Criação e recuperação de **UITableViewCell**s

Neste capítulo, cada célula exibirá o `description` de uma `BNRItem` como sua `textLabel`. Para isso, você precisa implementar o segundo método obrigatório do protocolo `UITableViewDataSource`, o `tableView:cellForRowAtIndexPath:`. Esse método cria uma célula, define sua `textLabel` como a `description` da `BNRItem` correspondente, e a retorna à `UITableView` (Figure 8.11).

Figure 8.11 Recuperação de UITableViewCell



Como você decide a que célula uma **BNRItem** corresponde? Um dos parâmetros enviados ao **tableView:cellForRowIndexPath:** é uma **NSIndexPath**, que tem duas propriedades: **section** e **row**. Quando essa mensagem é enviada a uma fonte de dados, a visão de tabela está pedindo: “Pode me dar uma célula para ser exibida na seção X, linha Y?”. Como só há uma seção neste exercício, a sua implementação se concentrará apenas na linha.

No **BNRItemsViewController.m**, implemente o **tableView:cellForRowIndexPath:**: de modo que a *enésima* linha exiba a *enésima* entrada do array **allItems**.

```
- (UITableViewCell *)tableView:(UITableView *)tableView
    cellForRowAtIndexPath:(NSIndexPath *)indexPath
{
    // Create an instance of UITableViewCell, with default appearance
    UITableViewCell *cell =
        [[UITableViewCell alloc] initWithStyle:UITableViewCellStyleDefault
            reuseIdentifier:@"UITableViewCell"];

    // Set the text on the cell with the description of the item
    // that is at the nth index of items, where n = row this cell
    // will appear in on the tableview
    NSArray *items = [[BNRItemStore sharedStore] allItems];
    BNRItem *item = items[indexPath.row];

    cell.textLabel.text = [item description];

    return cell;
}
```

Compile e execute o aplicativo agora e você verá uma **UITableView** preenchida com uma lista de itens aleatórios.

Pense novamente no seu projeto RandomItems do Chapter 3. Você criou a classe **BNRItem**, criou instâncias de **BNRItem** (objetos de modelo) e imprimiu seus dados no console.

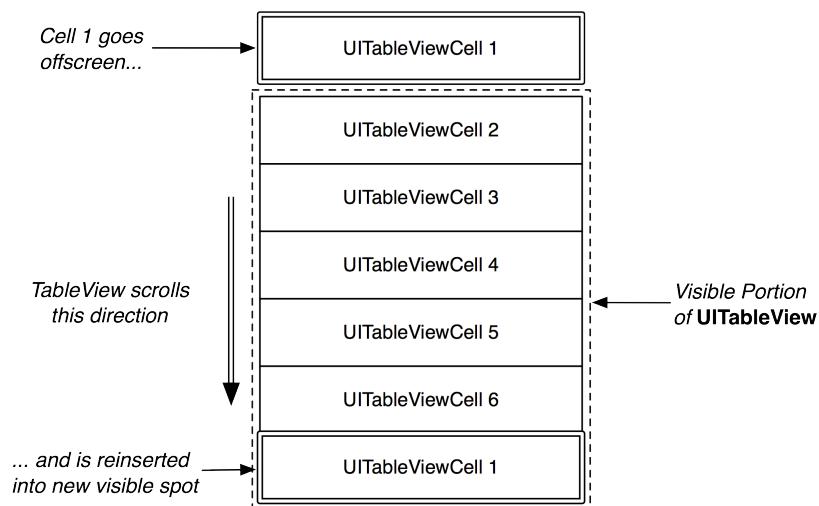
Agora você está reutilizando essa classe com um controlador diferente e deixando a interface do controlador com uma visão diferente. Você não teve que mudar nada da **BNRItem**, mas agora você pode mostrar seus dados de uma maneira completamente diferente. Esse é o padrão Modelo-Visão-Controlador em ação. Se você projetar suas classes e aplicativos de acordo com o MVC, será muito mais simples reutilizar essas classes em diferentes contextos.

## Reutilização de UITableViewCells

Dispositivos iOS têm uma quantidade limitada de memória. Se você estivesse exibindo uma lista com milhares de entradas em uma **UITableView**, teria milhares de instâncias de **UITableViewCell**. E o seu sofrido iPhone sucumbiria e morreria. Em seu último suspiro, ele diria: “Você só precisava de células suficientes para preencher a tela!”. E estaria certo.

Para preservar a vida dos dispositivos iOS, você pode reutilizar células da visão de tabela. Quando o usuário faz a rolagem da tabela, algumas células desaparecem da tela. As células fora da tela são colocadas em um pool (depósito) de células disponíveis para reutilização. Assim, em vez de criar uma célula totalmente nova para cada solicitação, a fonte de dados primeiro consulta esse depósito. Se houver uma célula não utilizada, a fonte de dados configura essa célula com novos dados e a retorna para a visão de tabela.

Figure 8.12 Instâncias reutilizáveis de **UITableViewCell**



Há um problema: às vezes, uma **UITableView** tem diferentes tipos de células. Ocasionalmente, você precisa ter subclasses de **UITableViewCell** para criar uma aparência ou um comportamento especial. Entretanto, diferentes subclasses flutuando em torno do depósito de células reutilizáveis criam a possibilidade de uma célula do tipo incorreto ser retornada. Você deve estar seguro do tipo de célula retornado para você para que você possa ter certeza das propriedades e dos métodos que ela tem.

Observe que não importa se você vai obter uma célula específica do depósito, porque você alterará o conteúdo dela de qualquer maneira. O que você precisa é de uma célula de um tipo específico. A boa notícia é que toda célula tem uma propriedade `reuseIdentifier` do tipo `NSString`. Quando uma fonte de dados solicita à visão de tabela uma célula reutilizável, ela passa uma string e diz: “Preciso de uma célula com esse identificador de reutilização”. Por convenção, o identificador de reutilização geralmente é o nome da classe da célula.

No `BNRItemsViewController.m`, atualize o `tableView:cellForRowAtIndexPath:` para reutilizar células:

```
- (UITableViewCell *)tableView:(UITableView *)tableView
    cellForRowAtIndexPath:(NSIndexPath *)indexPath
{
    UITableViewCell *cell =
        [[UITableViewCell alloc] initWithStyle:UITableViewCellStyleDefault
            reuseIdentifier:@"UITableViewCell"];

    // Get a new or recycled cell
    UITableViewCell *cell =
        [tableView dequeueReusableCellWithIdentifier:@"UITableViewCell"
            forIndexPath:indexPath];

    NSArray *items = [[BNRItemStore sharedStore] allItems];
    BNRItem *item = items[indexPath.row];

    cell.textLabel.text = [item description];

    return cell;
}
```

Anteriormente, você criou a célula da visão de tabela de forma explícita, mas agora você dará esse controle para a Apple para obter os benefícios do identificador de reutilização. Para que isso funcione, você deve informar à visão de tabela que tipo de célula ela deve instanciar se não houver células no depósito de reutilização.

No `BNRItemsViewController.m`, sobrescreva o `viewDidLoad` para registrar a classe `UITableViewCell` com a visão de tabela.

```
- (void)viewDidLoad
{
    [super viewDidLoad];

    [self.tableView registerClass:[UITableViewCell class]
        forCellReuseIdentifier:@"UITableViewCell"];
}
```

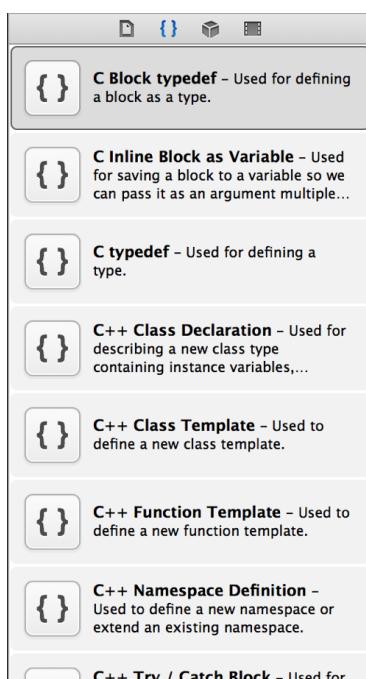
A reutilização de células significa que você só precisa criar algumas células, e isso diminui a demanda de memória. Os usuários do seu aplicativo (e os dispositivos deles) agradecerão. Compile e execute o aplicativo. O comportamento do aplicativo deve permanecer igual.

## Biblioteca de fragmentos de código

Talvez você tenha notado que quando começa a digitar a definição de método para o `init` em um arquivo de implementação, o Xcode adiciona automaticamente uma implementação de `init` no seu arquivo-fonte. Se não percebeu, então digite `init` em um arquivo de implementação e espere que o preenchimento de código comece a funcionar.

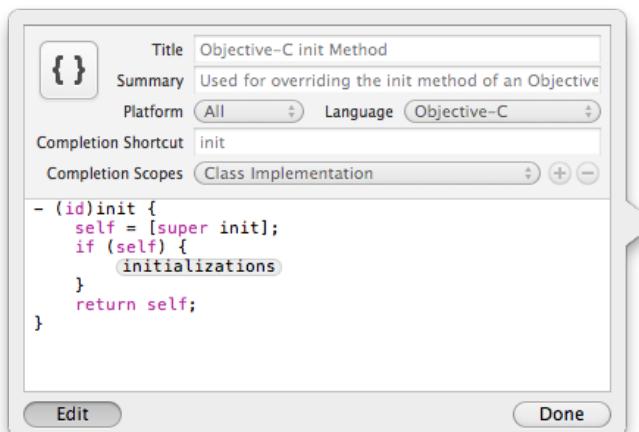
Esse código de cortesia vem da *biblioteca de fragmentos de código*. Para ver a biblioteca de fragmentos de código, abra a área de utilitários e selecione a guia {} no seletor de bibliotecas (Figure 8.13). Ou então, você pode usar o atalho Command-Control-Option-2, que revela a área de utilitários e a Code Snippet Library. Troque o número no atalho para selecionar a biblioteca correspondente.

Figure 8.13 Biblioteca de fragmentos de código



Observe que há vários fragmentos de código disponíveis (Figure 8.13). Clique em um deles e uma janela será exibida com os detalhes daquele fragmento. Clique no botão Edit na janela de detalhes do fragmento de código (Figure 8.14).

Figure 8.14 Janela de edição de fragmento



O campo Completion Shortcut na janela de edição mostra a você o que digitar em um arquivo-fonte para que o Xcode adicione o fragmento. Essa janela também informa que esse fragmento pode ser usado em um arquivo do Objective-C, contanto que você esteja trabalhando dentro do escopo de uma implementação de classe.

Você não pode editar nenhum dos fragmentos de código predefinidos, mas pode criar os seus próprios. No `BNRItemsViewController.m`, localize a implementação do `tableView:numberOfRowsInSection:`. Realce todo o método:

```
- (NSInteger)tableView:(UITableView *)tableView
 numberOfRowsInSection:(NSInteger)section
{
    return [[[BNRItemStore sharedStore] allItems] count];
}
```

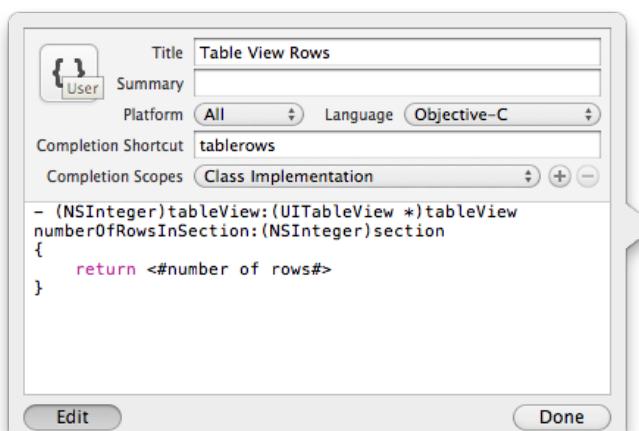
Arraste o código realçado para a biblioteca de fragmentos de código. A janela de edição aparece novamente, permitindo que você preencha os detalhes para esse fragmento.

Um problema com esse fragmento é que a instrução de retorno é realmente específica para esse aplicativo; seria muito mais útil se o valor retornado fosse um placeholder de preenchimento de código que você pudesse preencher facilmente. Na janela de edição, modifique o fragmento de código para que ele fique com a seguinte aparência:

```
- (NSInteger)tableView:(UITableView *)tableView
 numberOfRowsInSection:(NSInteger)section
{
    return <#number of rows#>;
}
```

Depois, preencha os outros campos na janela de edição, conforme mostrado na Figure 8.15 e clique em Done.

Figure 8.15 Criação de um novo fragmento



No `BNRItemsViewController.m`, comece a digitar `tablerows`. O Xcode recomendará esse fragmento de código e, se você pressionar a tecla de retorno, ele preencherá o código automaticamente para você, e o placeholder de número de linhas será selecionado. (Se houver vários placeholders, Control-/ fará você pular para o próximo.)

Antes de prosseguir, certifique-se de remover o código inserido pelo fragmento porque você já definiu o `tableView:numberOfRowsInSection:` no `BNRItemsViewController.m`.

## Desafio de bronze: seções

Faça a **UITableView** exibir duas seções: uma para itens que valem mais que \$ 50 e uma para todos os outros. Antes de iniciar este desafio, copie a pasta que contém o projeto e todos os arquivos-fonte no Finder. Depois, resolva o desafio na cópia do projeto; você precisará do original para os próximos capítulos.

## Desafio de prata: linhas constantes

Faça com que a última linha da **UITableView** sempre apresente o texto No more items!. Certifique-se de que essa linha sempre apareça, independentemente do número de itens no armazenamento (inclusive 0 item).

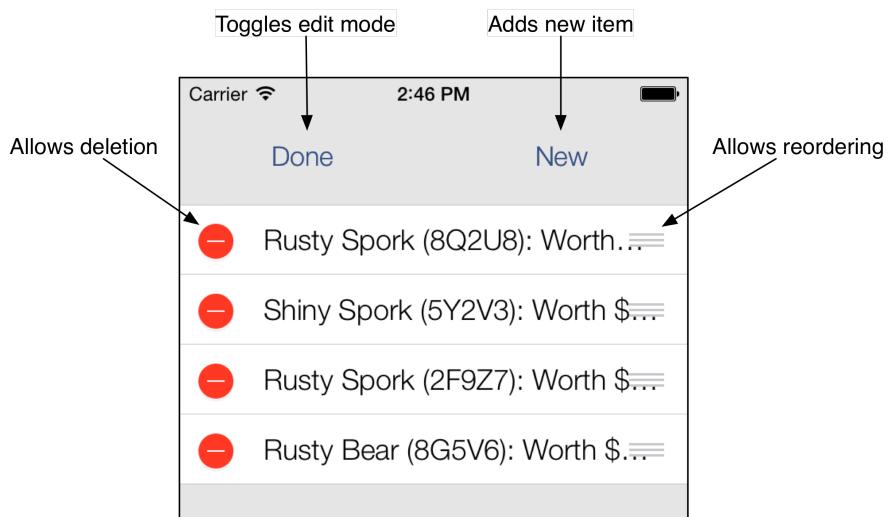
## Desafio de ouro: personalizar a tabela

Defina a altura de cada linha como 60 pontos, exceto a última linha do desafio de prata, que deve continuar com 44 pontos. Depois, altere o tamanho da fonte de todas as linhas, exceto a última, para 20 pontos. Finalmente, faça o plano de fundo da **UITableView** exibir uma imagem. (Para que fique tudo perfeito em termos de pixel, você precisará de uma imagem do tamanho correto, dependendo do dispositivo. Consulte a tabela no Chapter 1.)

# Edição de UITableView

No último capítulo, você criou um aplicativo que exibia uma lista de instâncias de `BNRItem` em uma `UITableView`. O próximo passo para o Homepwner é permitir a interação do usuário com a tabela – para adicionar, excluir e mover as linhas. A Figure 9.1 mostra como ficará o Homepwner ao final deste capítulo.

Figure 9.1 Homepwner no modo de edição



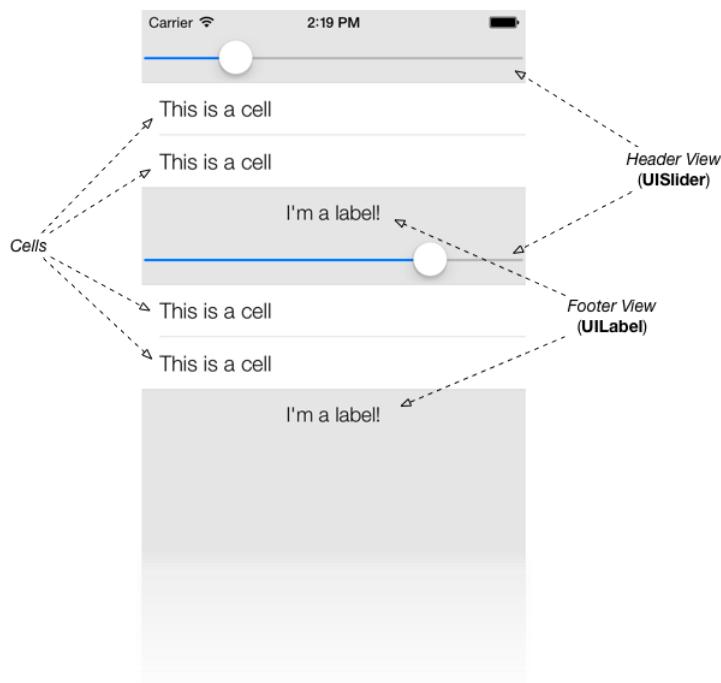
## Modo de edição

`UITableView` tem uma propriedade `editing`, e quando essa propriedade é configurada para `YES`, a `UITableView` entra no modo de edição. Quando a visão da tabela está no modo de edição, as linhas da tabela podem ser manipuladas pelo usuário. Dependendo de como a visão de tabela está configurada, o usuário pode alterar a ordem das linhas, acrescentar linhas ou remover linhas. O modo de edição não permite que o usuário edite o conteúdo da linha.

Mas antes, o usuário precisa de uma maneira de colocar a `UITableView` no modo de edição. Por enquanto, você vai incluir um botão que ativa o modo de edição na *visão de cabeçalho* da tabela. A visão de cabeçalho aparece no topo de uma tabela, e serve para adicionar títulos e controles que servirão para toda a seção ou toda a tabela. Pode ser qualquer instância de `UIView`.

Repare que a visão de tabela usa a palavra “cabeçalho” de duas maneiras diferentes: Pode haver um cabeçalho de tabela e podem haver cabeçalhos de seção. Da mesma forma, pode haver um rodapé de tabela e rodapés de seção.

Figure 9.2 Cabeçalhos e rodapés de seção



Você está criando uma visão de cabeçalho de tabela. Ela terá duas subvisões que são instâncias de **UIButton**: uma para ativar o modo de edição e outra para adicionar uma nova **BNRItem** à tabela. Você poderá criar esta visão de forma programática, mas, neste caso, criará a visão e suas subvisões em um arquivo XIB, e a **BNRItemsViewController** irá desarquivar esse arquivo XIB quando precisar mostrar a visão de cabeçalho.

Primeiro, vamos montar o código necessário. Abra novamente o `Homepwner.xcodeproj`. No `BNRItemsViewController.m`, acrescente uma extensão de classe com a seguinte propriedade. Além disso, cancele dois métodos na implementação.

```
@interface BNRItemsViewController : UITableViewController

@property (nonatomic, strong) IBOutlet UIView *headerView;

@end

@implementation BNRItemsViewController

// Other methods here

- (IBAction)addNewItem:(id)sender
{
}

- (IBAction)toggleEditingMode:(id)sender
{
}
```

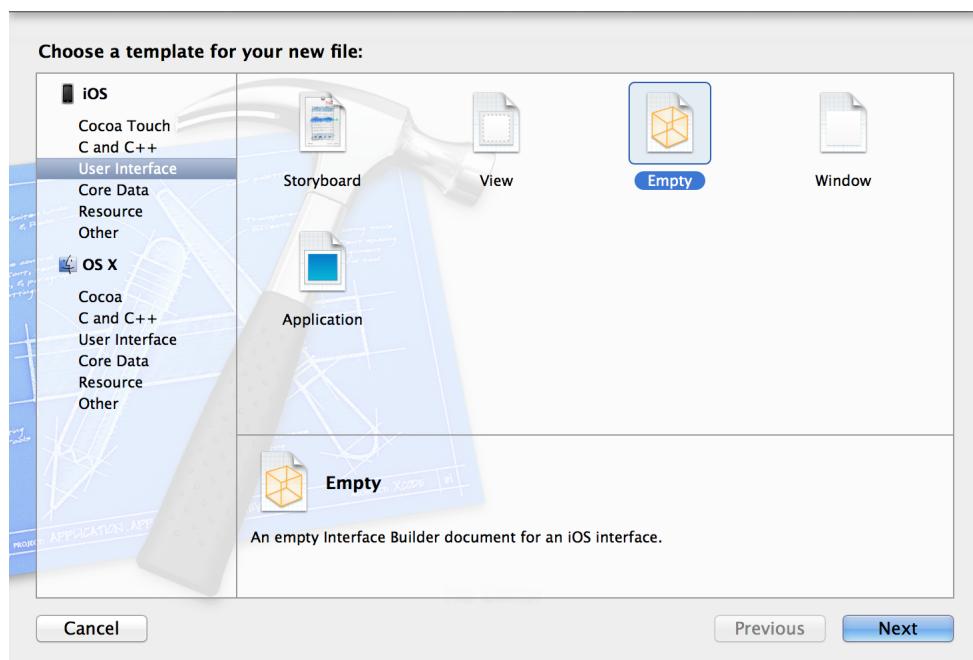
Observe que `headerView` é uma propriedade forte. Isso se deve ao fato de que ela será um objeto de nível superior no arquivo XIB; você usa referências fracas para objetos que são de propriedade (direta ou indiretamente) dos objetos de nível superior.

Agora, você precisa criar o novo arquivo XIB. Diferentemente dos arquivos XIB anteriores que você criou, este arquivo XIB não lidará com a view do controlador de visão. (Como uma subclasse de **UITableViewController**, **BNRItemsViewController** já sabe como criar sua view.) Os arquivos XIB, geralmente, são usados para criar

uma visão para um controlador de visão, mas também podem ser usados sempre que você quiser dispor os objetos de visão, arquivá-los ou carregá-los no tempo de execução.

Crie um novo arquivo (Command-N): Na seção iOS, selecione User Interface, escolha o template Empty e clique em Next (Figure 9.3).

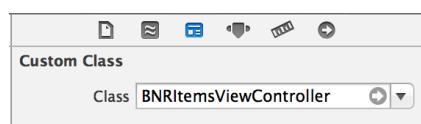
Figure 9.3 Criação de um novo arquivo XIB



No painel seguinte, selecione iPhone. Salve esse arquivo como HeaderView.

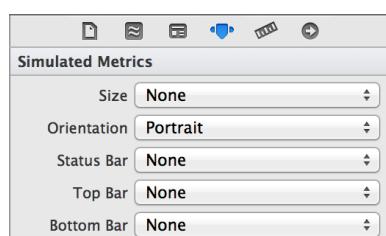
No HeaderView.xib, selecione o objeto File's Owner e altere sua Class para **BNRItemsViewController** no inspetor de identidade (Figure 9.4).

Figure 9.4 Alteração do File's Owner



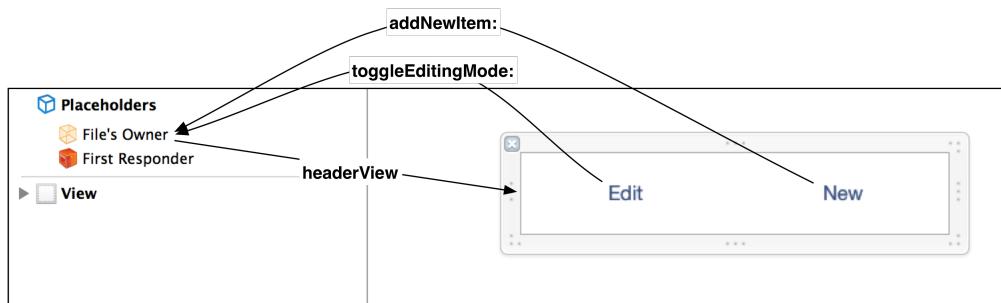
Arraste uma **UIView** para o canvas. Depois arraste duas instâncias de **UIButton** para essa visão. Agora, você irá querer redimensionar a **UIView** para que ela se encaixe exatamente nos botões; no entanto, o Xcode não permitirá isso: o tamanho está bloqueado. Para desbloquear o tamanho, selecione a **UIView** no canvas e abra o inspetor de atributos. Na seção Simulated Metrics, selecione None para a opção Size (Figure 9.5).

Figure 9.5 Desbloqueio do tamanho de uma visão



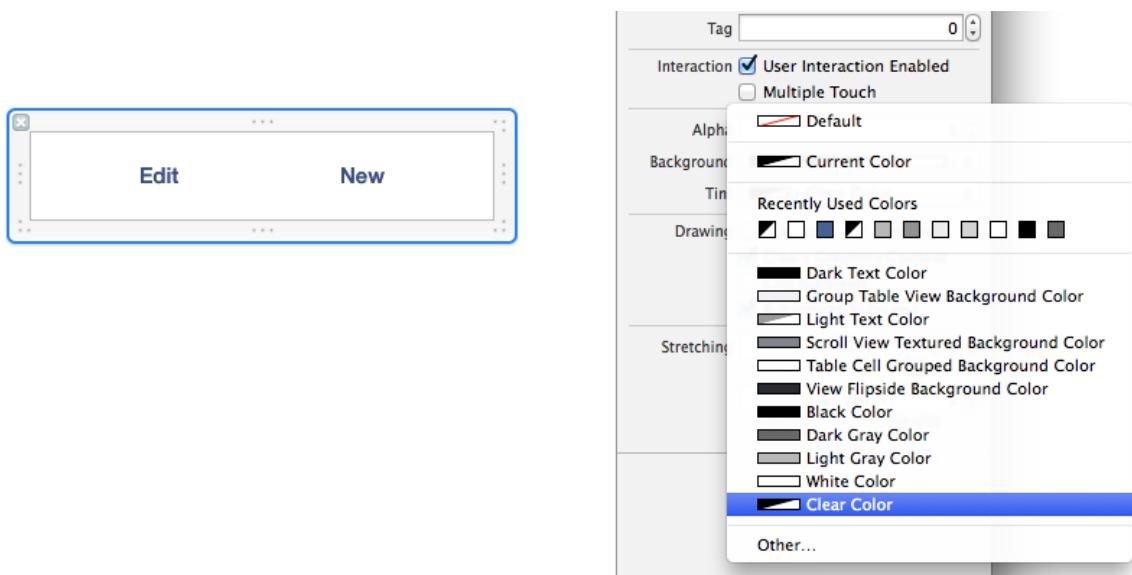
Agora que a visão pode ser redimensionada, redimensione-a e faça as conexões mostradas em Figure 9.6.

Figure 9.6 Disposição da HeaderView.xib



Além disso, altere a cor do fundo da instância de **UIView** para que fique completamente transparente. Para fazer isso, selecione a visão e exiba o inspetor de atributos. No pop-up identificado como Background, escolha Clear Color (Figure 9.7).

Figure 9.7 Definição de transparente para cor de fundo



Até aqui, os seus arquivos XIB foram carregados automaticamente com a implementação de **UIViewController**. Por exemplo, **BNRReminderViewController** na Chapter 6 sabia como carregar o **BNRReminderViewController.xib** graças ao código escrito em sua superclasse, **UIViewController**. Para o **HeaderView.xib**, você terá que escrever o código de modo que **BNRItemsViewController** carregue esse arquivo XIB manualmente.

Para carregar manualmente um arquivo XIB, você usa **NSBundle**. Essa classe é a interface entre um aplicativo e o pacote de aplicativos em que reside. Quando quiser acessar um arquivo em um pacote de aplicativos, você deve solicitá-lo à **NSBundle**. Uma instância de **NSBundle** é criada quando o seu aplicativo é aberto, e você pode obter um ponteiro para essa instância, enviando a mensagem **mainBundle** para **NSBundle**.

Depois de obter um ponteiro para o principal objeto do pacote, você pode pedir que ele carregue um arquivo XIB. No **BNRItemsViewController.m**, implemente **headerView**.

```
- (UIView *)headerView
{
    // If you have not loaded the headerView yet...
    if (!_headerView) {

        // Load HeaderView.xib
        [[NSBundle mainBundle] loadNibNamed:@"HeaderView"
                                      owner:self
                                      options:nil];
    }

    return _headerView;
}
```

Repare que este é um método getter que faz mais do que apenas obter. Este é um padrão comum: *Lazy Instantiation* adia a criação do objeto até que ela seja necessária de fato. Em alguns casos, essa abordagem pode reduzir显著mente o espaço ocupado na memória pelo seu aplicativo.

Você não precisa especificar o sufixo do nome do arquivo; **NSBundle** o deduzirá. Além disso, observe que você passou **self** como proprietária do arquivo XIB. Isso garante que, quando a **NSBundle** principal estiver analisando sintaticamente o arquivo NIB resultante no tempo de execução, quaisquer conexões ao placeholder do File's Owner serão feitas a essa instância de **BNRItemsViewController**.

Na primeira vez em que a mensagem **headerView** é enviada à **BNRItemsViewController**, ela carrega o **HeaderView.xib** e mantém um ponteiro para o objeto de visão na variável de instância **headerView**. Os botões nessa visão irão enviar mensagens à **BNRItemsViewController** quando tocados.

Agora, você só precisa informar à visão de tabela sua visão de cabeçalho. No **BNRItemsViewController.m**, acrescente isto ao método **viewDidLoad**:

```
- (void)viewDidLoad
{
    [super viewDidLoad];

    [self.tableView registerClass:[UITableViewCell class]
        forCellReuseIdentifier:@"UITableViewCell"];

    UIView *header = self.headerView;
    [self.tableView setTableHeaderView:header];
}
```

Compile e execute o aplicativo para ver a interface.

Por mais que os arquivos XIB sejam normalmente usados para criar a visão de um controlador de visão (por exemplo, o **BNRReminderViewController.xib**), você acaba de ver que um arquivo XIB pode ser usado toda vez que você quiser arquivar objetos de visão. Além disso, qualquer objeto pode carregar manualmente um arquivo XIB, enviando a mensagem **loadNibName:owner:options:** ao pacote de aplicativos.

O comportamento padrão de carregamento de XIB da **UIViewController** usa o mesmo código. A única diferença é que ele conecta o outlet de sua **view** ao objeto de visão no arquivo XIB. Imagine como provavelmente seria a implementação padrão do **loadView** para **UIViewController**:

```

- (void)loadView
{
    // Which bundle is the NIB in?
    // Was a bundle passed to initWithNibName:bundle:?
    NSBundle *bundle = [self nibBundle];
    if (!bundle) {

        // Use the default
        bundle = [NSBundle mainBundle];
    }

    // What is the NIB named?
    // Was a name passed to initWithNibName:bundle:?
    NSString *nibName = [self nibName];
    if (!nibName) {

        // Use the default
        nibName = NSStringFromClass([self class]);
    }

    // Try to find the NIB in the bundle
    NSString *nibPath = [bundle pathForResource:nibName
                                    ofType:@"nib"];

    // Does it exist?
    if (nibPath) {

        // Load it (this will set the view outlet as a side-effect
        [bundle loadNibNamed:nibName owner:self options:nil];
    } else {

        // If there is no NIB, just create a blank UIView
        self.view = [[UIView alloc] init];
    }
}

```

Agora, vamos implementar o método `toggleEditMode:`. Você poderia ativar a propriedade `editing` de `UITableView` diretamente. No entanto, `UIViewController` também tem uma propriedade `editing`. Uma instância de `UIViewController` define automaticamente a propriedade `editing` de sua visão de tabela para corresponder à sua própria propriedade `editing`.

Para definir a propriedade `editing` de um controlador de visão, você envia a ele a mensagem `setEditing:animated:`. No `BNRItemsViewController.m`, implemente `toggleEditMode:`.

```

- (IBAction)toggleEditMode:(id)sender
{
    // If you are currently in editing mode...
    if (self.isEditing) {

        // Change text of button to inform user of state
        [sender setTitle:@"Edit" forState:UIControlStateNormal];

        // Turn off editing mode
        [self setEditing:NO animated:YES];
    } else {

        // Change text of button to inform user of state
        [sender setTitle:@"Done" forState:UIControlStateNormal];

        // Enter editing mode
        [self setEditing:YES animated:YES];
    }
}

```

Compile e execute seu aplicativo, toque no botão Edit e a `UITableView` entrará no modo de edição (Figure 9.8).

Figure 9.8 **UITableView** no modo de edição

## Adição de linhas

Há duas interfaces comuns para adicionar linhas a uma visão de tabela no tempo de execução.

- Um botão acima das células da visão de tabela. Isso costuma servir para acrescentar um registro para o qual haja uma visão detalhada. Por exemplo, no aplicativo **Contacts**, você toca em um botão quando conhece uma pessoa nova e quer anotar todas as informações dela.
- Uma célula com um sinal de mais verde. Ela costuma servir para adicionar um novo campo a um registro, como quando você quer adicionar um aniversário ao registro de uma pessoa no aplicativo **Contacts**. No modo de edição, você toca no sinal de mais verde ao lado de “add birthday”.

Neste exercício, você usa o botão **New** na visão de cabeçalho, em vez disso. Quando esse botão é tocado, uma nova linha é adicionada à **UITableView**.

No **BNRItemsViewController.m**, implemente **addNewItem:**.

```
- (IBAction)addNewItem:(id)sender
{
    // Make a new index path for the 0th section, last row
    NSInteger lastRow = [self.tableView numberOfRowsInSection:0];
    NSIndexPath *indexPath = [NSIndexPath indexPathForRow:lastRow inSection:0];

    // Insert this new row into the table.
    [self.tableView insertRowsAtIndexPaths:@[indexPath]
                           withRowAnimation:UITableViewRowAnimationTop];
}
```

Compile e execute o aplicativo. Toque no botão **New** e... o aplicativo trava. O console lhe diz que a visão de tabela tem uma exceção de inconsistência interna.

Lembre-se de que, no final das contas, é a **dataSource** da **UITableView** que determina o número de linhas que a visão de tabela vai exibir. Após inserir uma nova linha, a visão de tabela terá seis linhas (as cinco originais mais a nova). Depois, ela retorna a sua **dataSource** e pergunta qual o número de linhas que deve exibir. A **BNRItemsViewController** consulta o armazenamento e retorna a informação de que deve haver cinco linhas. Então, a **UITableView** diz: “Peraí, tem alguma coisa errada!” e dispara uma exceção.

Você precisa ter certeza de que a **UITableView** e sua **dataSource** estejam de acordo em relação ao número de linhas. Assim, você deve adicionar também uma nova **BNRItem** à **BNRItemStore** antes de inserir a nova linha.

No **BNRItemsViewController.m**, atualize o **addNewItem:**.

```

- (IBAction)addNewItem:(id)sender
{
    NSInteger lastRow = [[self tableView] numberOfRowsInSection:0];

    // Create a new BNRItem and add it to the store
    BNRItem *newItem = [[BNRItemStore sharedStore] createItem];

    // Figure out where that item is in the array
    NSInteger lastRow = [[[BNRItemStore sharedStore] allItems] indexOfObject:newItem];

    NSIndexPath *ip = [NSIndexPath indexPathForRow:lastRow inSection:0];

    // Insert this new row into the table
    [self.tableView insertRowsAtIndexPaths:@[ip]
        withRowAnimation:UITableViewRowAnimationTop];
}

```

Compile e execute o aplicativo. Toque no botão New e observe enquanto uma nova linha aparece na parte inferior da tabela. Lembre-se de que a função de uma visão de objeto é apresentar os objetos de modelo ao usuário; atualizar as visões sem atualizar os objetos de modelo não tem muita utilidade.

Observe também que você está enviando a mensagem `tableView` à `BNRItemsViewController` para chegar na visão de tabela. Esse método é herdado da `UITableViewController`, e retorna a visão de tabela do controlador. Embora você possa enviar a mensagem `view` a uma instância de `UITableViewController` e obter um ponteiro para o mesmo objeto, quando você usa `tableView`, diz ao compilador que o objeto retornado será uma instância da classe `UITableView`. Portanto, o envio de uma mensagem que seja específica à `UITableView`, como `insertRowsAtIndexPaths:withRowAnimation:`, não gera nenhum aviso.

Agora que você sabe adicionar linhas e itens, remova o código do método `init` no `BNRItemsViewController.m` que coloca 5 itens aleatórios no armazenamento.

```

- (instancetype)init
{
    // Call the superclass's designated initializer
    self = [super initWithStyle:UITableViewStylePlain];
    if (self) {
        for (int i = 0; i < 5; i++) {
            [[BNRItemStore sharedStore] createItem];
        }
    }
    return self;
}

```

Compile e execute o aplicativo. Não haverá nenhuma linha quando abrir o aplicativo pela primeira vez, mas você pode adicionar algumas, tocando no botão New.

## Exclusão de linhas

No modo de edição, os círculos vermelhos com o sinal de menos (mostrados na Figure 9.8) são controles de exclusão, e, ao tocá-los, você exclui a linha em questão. No entanto, por enquanto, nada acontece quando você toca no controle de exclusão. (Tente e veja você mesmo.) Antes de a visão de tabela excluir uma linha, ela envia a sua fonte de dados uma mensagem sobre a exclusão desejada, e espera uma mensagem de confirmação antes de executar a ação.

Para excluir uma célula, você precisa fazer duas coisas: remover a linha de `UITableView` e remover a `BNRItem` associada a ela da `BNRItemStore`. Para isso, `BNRItemStore` precisa saber como remover objetos de si própria.

No `BNRItemStore.h`, declare um novo método.

```

@interface BNRItemStore : NSObject
+ (BNRItemStore *)sharedStore;

@property (nonatomic, strong, readonly) NSArray *allItems;

- (BNRItem *)createItem;
- (void)removeItem:(BNRItem *)item;

@end

```

No `BNRItemStore.m`, implemente `removeItem:`:

```
- (void)removeItem:(BNRItem *)item
{
    [self.privateItems removeObjectIdenticalTo:item];
}
```

Você poderia usar o método `NSMutableArray` de `removeObject`: aqui em vez do `removeObjectIdenticalTo`: mas veja a diferença: O `removeObject`: vai em cada um dos objetos no array e envia a ele a mensagem `isEqual`: Uma classe pode implementar esse método para retornar YES ou NO com base em sua própria determinação. Por exemplo, dois objetos `BNRItem` podem ser considerados iguais quando possuem a mesma `valueInDollars`.

O método `removeObjectIdenticalTo`: por outro lado, remove um objeto se, e somente se, for exatamente o mesmo objeto que foi passado nessa mensagem. Embora `BNRItem` atualmente não sobrescreva `isEqual`: para verificações especiais, talvez esse seja o caso no futuro. Portanto, você deve usar `removeObjectIdenticalTo`: quando estiver especificando uma instância em particular.

Agora, você implementará `tableView:commitEditingStyle:forRowAtIndexPath`: um método do protocolo `UITableViewDataSource`. (Essa mensagem é enviada à `BNRItemsViewController`. Tenha em mente que, embora seja na `BNRItemStore` que os dados são mantidos, o `BNRItemsViewController` é a `dataSource` da visão de tabela.)

Quando `tableView:commitEditingStyle:forRowAtIndexPath`: é enviado para a fonte de dados, dois argumentos extras são passados juntamente com ele. O primeiro é a `UITableViewCellEditingStyle` que, nesse caso, é `UITableViewCellEditingStyleDelete`. O outro argumento é a `NSIndexPath` da linha na tabela.

No `BNRItemsViewController.m`, implemente esse método para que `BNRItemStore` remova o objeto correto, e confirme a exclusão da linha enviando a mensagem `deleteRowsAtIndexPaths:withRowAnimation`: de volta para a visão de tabela.

```
- (void)tableView:(UITableView *)tableView
    commitEditingStyle:(UITableViewCellEditingStyle)editingStyle
    forRowAtIndexPath:(NSIndexPath *)indexPath
{
    // If the table view is asking to commit a delete command...
    if (editingStyle == UITableViewCellEditingStyleDelete) {
        NSArray *items = [[BNRItemStore sharedStore] allItems];
        BNRItem *item = items[indexPath.row];
        [ps removeItem:item];

        // Also remove that row from the table view with an animation
        [tableView deleteRowsAtIndexPaths:@[indexPath]
                           withRowAnimation:UITableViewRowAnimationFade];
    }
}
```

Compile e execute o seu aplicativo, crie algumas linhas e depois exclua uma delas. Ela deverá desaparecer. Observe que arrastar-para-excluir também funciona.

## Movimentação de linhas

Para alterar a ordem das linhas em uma `UITableView`, você usará outro método do protocolo `UITableViewDataSource` – `tableView:moveRowAtIndexPath:toIndexPath`:

Para excluir uma linha, você precisou enviar a mensagem `deleteRowsAtIndexPaths:withRowAnimation`: à `UITableView` para confirmar a exclusão. Para mover uma linha, entretanto, não é necessário confirmar; a visão de tabela move a linha com base em sua própria autoridade e reporta a movimentação à sua fonte de dados enviando a mensagem `tableView:moveRowAtIndexPath:toIndexPath`:. Você só precisa implementar esse método para atualizar sua fonte de dados para combinar com a nova ordem.

Mas, antes de poder implementar o método da fonte de dados, você precisa dar à `BNRItemStore` um método que altere a ordem dos itens em seu array `allItems`. No `BNRItemStore.h`, declare esse método.

```
- (void)moveItemAtIndex:(int)fromIndex
    toIndex:(int)toIndex;
```

No `BNRItemStore.m`, implemente `moveItemAtIndex:toIndex`:

```

- (void)moveItemAtIndexPath:(NSInteger)fromIndex
    toIndexPath:(NSInteger)toIndex
{
    if (fromIndex == toIndex) {
        return;
    }
    // Get pointer to object being moved so you can re-insert it
    BNRItem *item = self.privateItems[fromIndex];

    // Remove item from array
    [self.privateItems removeObjectAtIndex:fromIndex];

    // Insert item in array at new location
    [self.privateItems insertObject:item atIndex:toIndex];
}

```

No `BNRItemsViewController.m`, implemente `tableView:moveRowAtIndexPath:toIndexPath:` para atualizar o armazenamento.

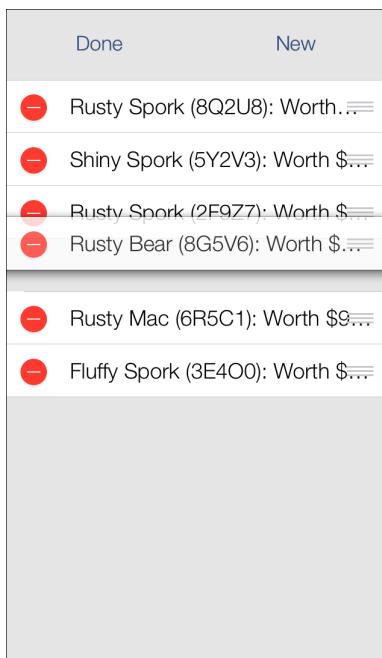
```

- (void)tableView:(UITableView *)tableView
    moveRowAtIndexPath:(NSIndexPath *)sourceIndexPath
    toIndexPath:(NSIndexPath *)destinationIndexPath
{
    [[BNRItemStore sharedStore] moveItemAtIndexPath:sourceIndexPath.row
                                              toIndex:destinationIndexPath.row];
}

```

Compile e execute o aplicativo. Observe os novos controles de reordenação (as três linhas horizontais) ao lado de cada linha. Pressione e segure um controle de reordenação, e mova a linha para outra posição (Figure 9.9).

Figure 9.9 Movimentação de uma linha



Observe que simplesmente implementar `tableView:moveRowAtIndexPath:toIndexPath:` fez com que os controles de reordenação aparecessem. A `UITableView` pode perguntar à sua fonte de dados no tempo de execução se ela implementa o `tableView:moveRowAtIndexPath:toIndexPath:`. Se sim, a visão de tabela diz: “Muito bem, então você consegue mover linhas. Vou adicionar os controles de reordenação”. Caso isso não aconteça, ele diz: “Se você não vai implementar esse método, não vou colocar controles ali.”

## Desafio de bronze: renomear o botão Delete

Quando se exclui uma linha, aparece um botão de confirmação identificado como Delete. Altere o nome desse botão para Remove.

## **Desafio de prata: impedir a reordenação**

Faça com que a visão de tabela mostre sempre uma última linha com os dizeres `No more items!` (esta parte é a mesma de um desafio do último capítulo. Se você já tinha feito, excelente!). Agora faça com que essa linha não possa ser movida.

## **Desafio de ouro: realmente impedir a reordenação**

Após concluir o desafio de prata, você talvez perceba que, embora não possa mover a linha `No more items!` propriamente dita, você ainda pode arrastar outras linhas para debaixo dela. Agora faça com que, não importa o que aconteça, a linha `No more items!` nunca possa sair da última posição na tabela.



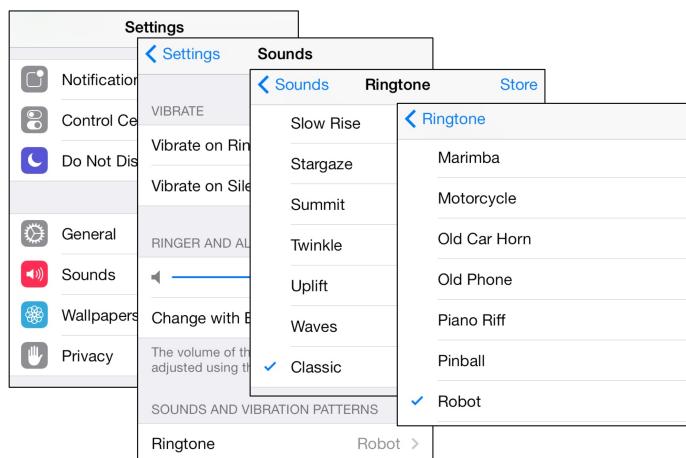
# 10

## UINavigationController

No Chapter 6, você aprendeu sobre **UITabBarController** e como ele permite que um usuário acesse telas diferentes. Um controlador de barra de guias é ótimo quando você tem telas independentes, mas se você quiser se mover entre telas relacionadas?

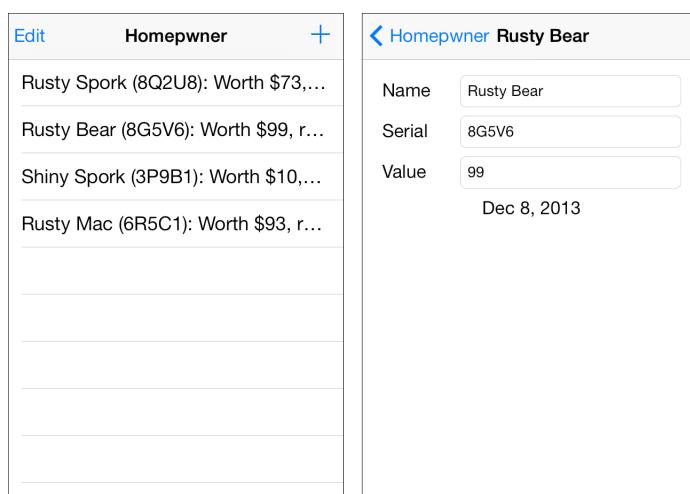
Por exemplo, o aplicativo **Settings** tem várias telas de informação relacionadas: uma lista de configurações (como Sounds), uma página detalhada para cada configuração e uma página de seleção para cada detalhe. Esse tipo de interface é chamada de *interface drill-down*.

Figure 10.1 Configurações com uma interface drill-down



Neste capítulo, você usará **UINavigationController** para adicionar uma interface drill-down ao Homeowner que permite que o usuário visualize e edite os detalhes de uma **BNRItem** (Figure 10.2).

Figure 10.2 Homeowner com **UINavigationController**



## UINavigationController

Quando o aplicativo apresenta várias telas de informações, **UINavigationController** mantém uma pilha dessas telas. Cada tela é a view de uma **UIViewController** e a pilha é um array de controladores de visão. Quando uma **UIViewController** encontra-se no topo da pilha, sua view é visível.

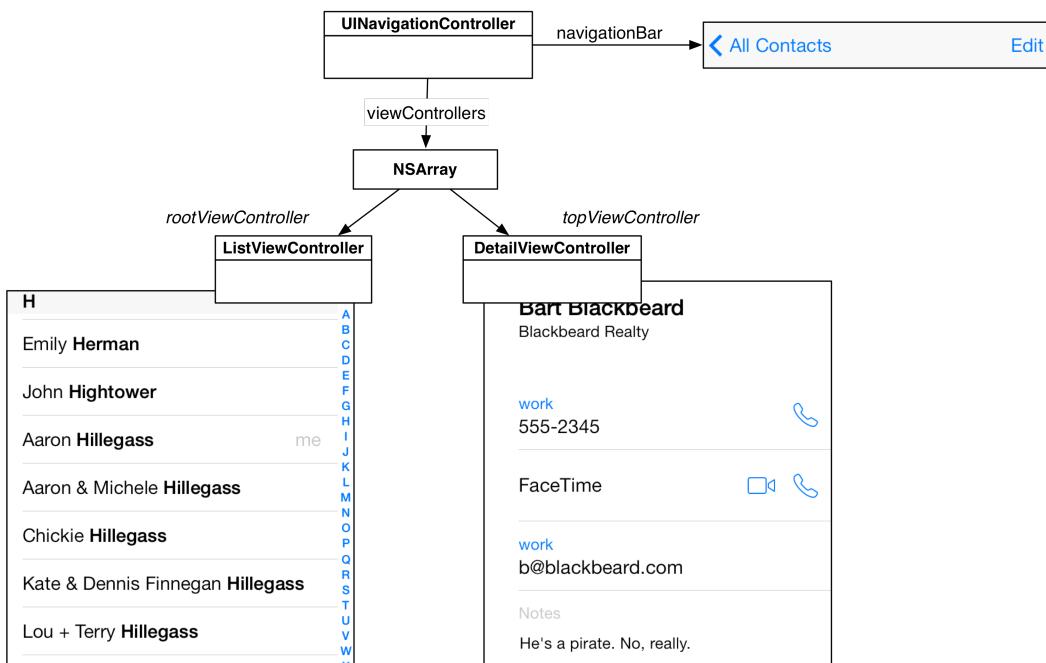
Quando você inicializa uma instância de **UINavigationController**, você dá a ela uma **UIViewController**.

Essa **UIViewController** é o *controlador de visão raiz* do controlador de navegação. O controlador de visão raiz está sempre no final da pilha. Mais controladores de visão podem ser enviados para o início da pilha de **UINavigationController**, enquanto o aplicativo é executado.

Quando uma **UIViewController** é enviada para a pilha, sua view desliza para a tela pelo lado direito. Quando a pilha é disparada, o controlador de visão do topo é retirado da pilha e a visão desliza para a direita, exibindo a visão do próximo controlador de visão na pilha.

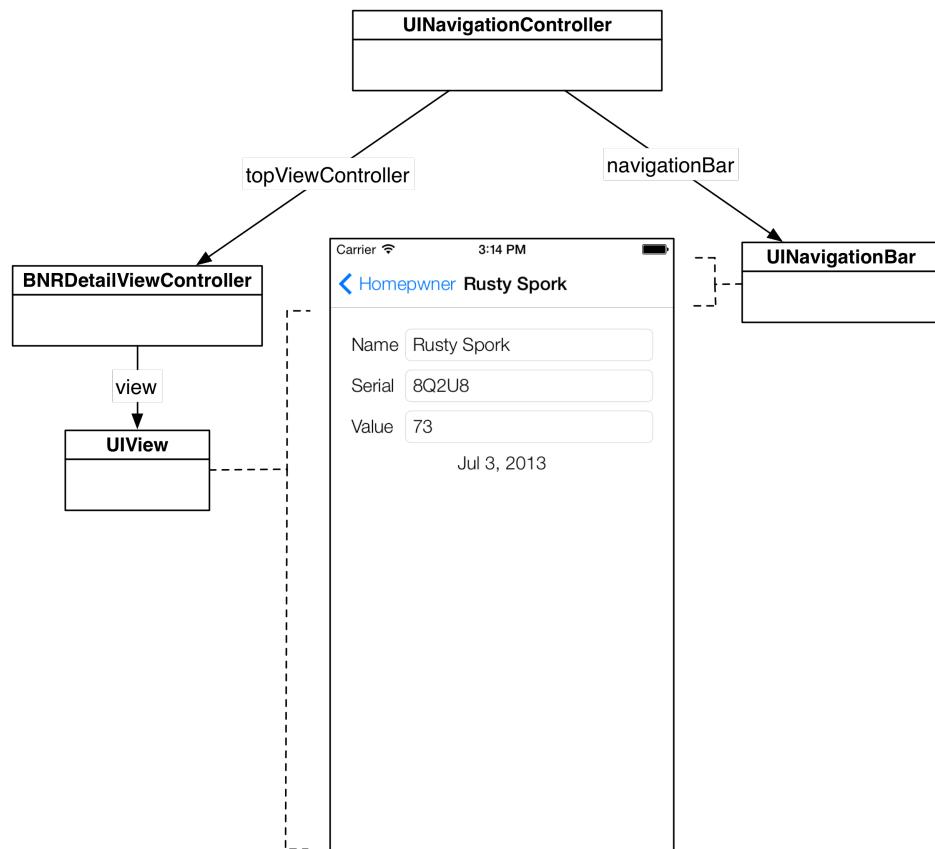
A Figure 10.3 mostra um controlador de navegação com dois controladores de visão: um controlador de visão raiz e um controlador de visão adicional acima dele, no topo da pilha. A view do controlador de visão adicional é o que o usuário vê, pois este controlador está no topo da pilha.

Figure 10.3 Pilha de **UINavigationController**



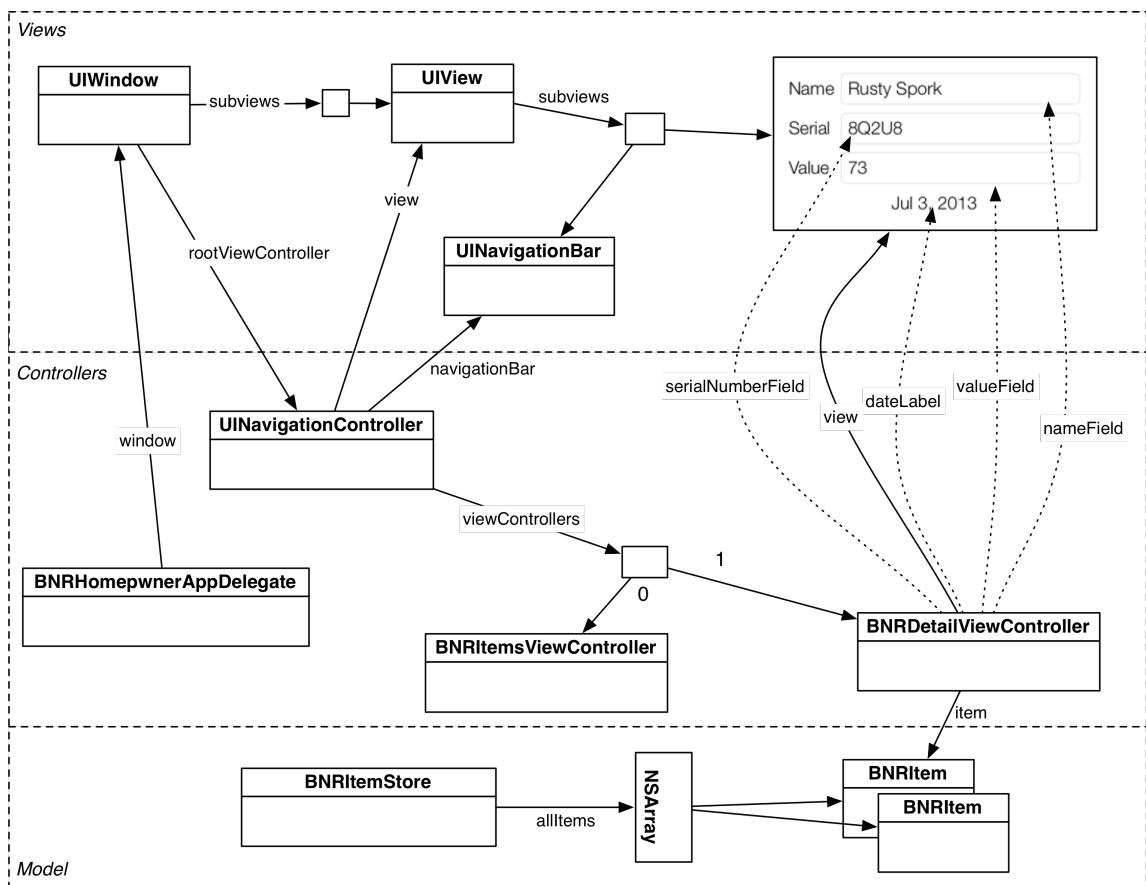
Como **UITabBarController**, **UINavigationController** tem um array de **viewControllers**. O controlador de visão raiz é o primeiro objeto no array. À medida que mais controladores de visão são enviados para a pilha, eles são adicionados ao final desse array. Assim, o último controlador de visão no array encontra-se no topo da pilha. A propriedade **topViewController** de **UINavigationController** mantém um ponteiro para o topo da pilha.

**UINavigationController** é uma subclasse de **UIViewController**, então, ela tem uma view própria. Sua view sempre tem duas subvisões: a **UINavigationBar** e a view de **topViewController** (Figure 10.4). Você pode definir um controlador de navegação como **rootViewController** da janela para tornar sua view uma subvisão da janela.

Figure 10.4 Uma visão de **UINavigationController**

Neste capítulo, você adicionará uma **UINavigationController** ao aplicativo Homepwner e tornará **BNRItemsViewController** a **UINavigationController** de **rootViewController**. Depois, você criará outra subclasse de **UIViewController** que pode ser enviada para a pilha de **UINavigationController**. Quando um usuário seleciona uma das setas, a nova visão de **UIViewController** deslizará para a tela. Esse controlador de visão permitirá que o usuário visualize e edite as propriedades da **BNRItem** selecionada. O diagrama de objetos para o aplicativo Homepwner atualizado é mostrado na Figure 10.5.

Figure 10.5 Diagrama de objetos do Homepwner



Este aplicativo está ficando bastante grande, como você pode ver no enorme diagrama de objetos. Felizmente, controladores de visão e **UINavigationController** sabem como lidar com esse tipo de diagrama complicado. Ao escrever aplicativos de iOS, é importante tratar cada **UIViewController** como seu próprio mundo. O que já foi implementado em Cocoa Touch fará o trabalho pesado.

Agora vamos dar ao Homepwner um controlador de navegação. Reabra o projeto do Homepwner e, em seguida, abra o `BNRAppDelegate.m`. Os únicos requisitos para usar uma **UINavigationController** são que você dê a ela um controlador de visão raiz e adicione sua view à janela.

No `BNRAppDelegate.m`, crie a **UINavigationController** em `application:didFinishLaunchingWithOptions:`, dê a ela seu próprio controlador de visão raiz e defina a **UINavigationController** como o controlador de visão raiz da janela.

```

- (BOOL)application:(UIApplication *)application
    didFinishLaunchingWithOptions:(NSDictionary *)launchOptions
{
    self.window = [[UIWindow alloc] initWithFrame:UIScreen.mainScreen.bounds];
    // Override point for customization after application launch

    BNRItemsViewController *itemsViewController
        = [[BNRItemsViewController alloc] init];

    // Create an instance of a UINavigationController
    // its stack contains only itemsViewController
    UINavigationController *navController = [[UINavigationController alloc]
        initWithRootViewController:itemsViewController];

    self.window.rootViewController = itemsViewController;
    // Place navigation controller's view in the window hierarchy
    self.window.rootViewController = navController;

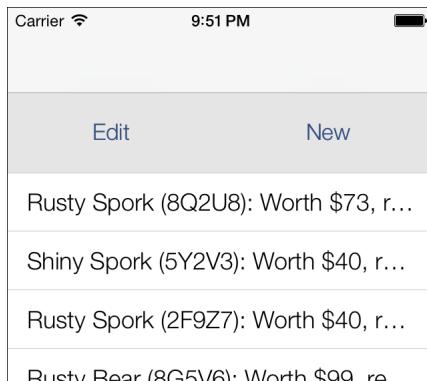
    self.window.backgroundColor = [UIColor whiteColor];
    [self.window makeKeyAndVisible];

    return YES;
}

```

Compile e execute o aplicativo. Homepwner terá a mesma aparência de antes – só que agora tem uma **UINavigationBar** na parte superior da tela (Figure 10.6). Observe como a **BNRItemsViewController** de view foi redimensionada para caber na tela com a barra de navegação. A **UINavigationController** fez isso por você.

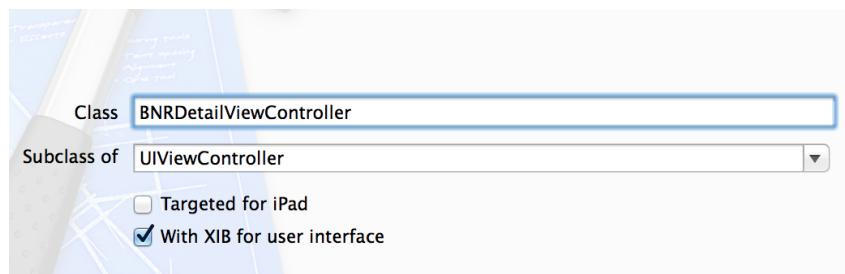
Figure 10.6 Homepwner com uma barra de navegação vazia



## UIViewController adicional

Para ver o poder real da classe **UINavigationController**, você precisa de outra **UIViewController** para colocar na pilha do controlador de navegação. Crie uma nova Objective-C class (File → New → File...). Nomeie essa classe como **BNRDetailViewController** e escolha UIViewController como a superclasse. Marque a caixa With XIB for user interface (Figure 10.7).

Figure 10.7 Criação da subclasse **UIViewController** com XIB



No `BNRDetailViewController.m`, exclua todos os códigos entre as diretivas `@implementation` e `@end`, de modo que o arquivo fique semelhante ao seguinte:

```
#import "BNRDetailViewController.h"

@interface BNRDetailViewController : UIViewController

@end

@implementation BNRDetailViewController
    // Implementation code goes here
@end
```

No `Homepwner`, você quer que o usuário seja capaz de tocar sobre um item e obter outra tela com campos de texto editáveis para cada propriedade daquela `BNRItem`. Essa visão será controlada por uma instância de `BNRDetailViewController`.

A visão de detalhes precisa de quatro subvisões – uma para cada variável de instância de uma instância `BNRItem`. E como você precisa ser capaz de acessar estas subvisões durante o tempo de execução, `BNRDetailViewController` precisa de outlets para elas. A ideia é adicionar quatro novos outlets à `BNRDetailViewController`, arrastar as subvisões para a visão do arquivo XIB e, em seguida, fazer as conexões.

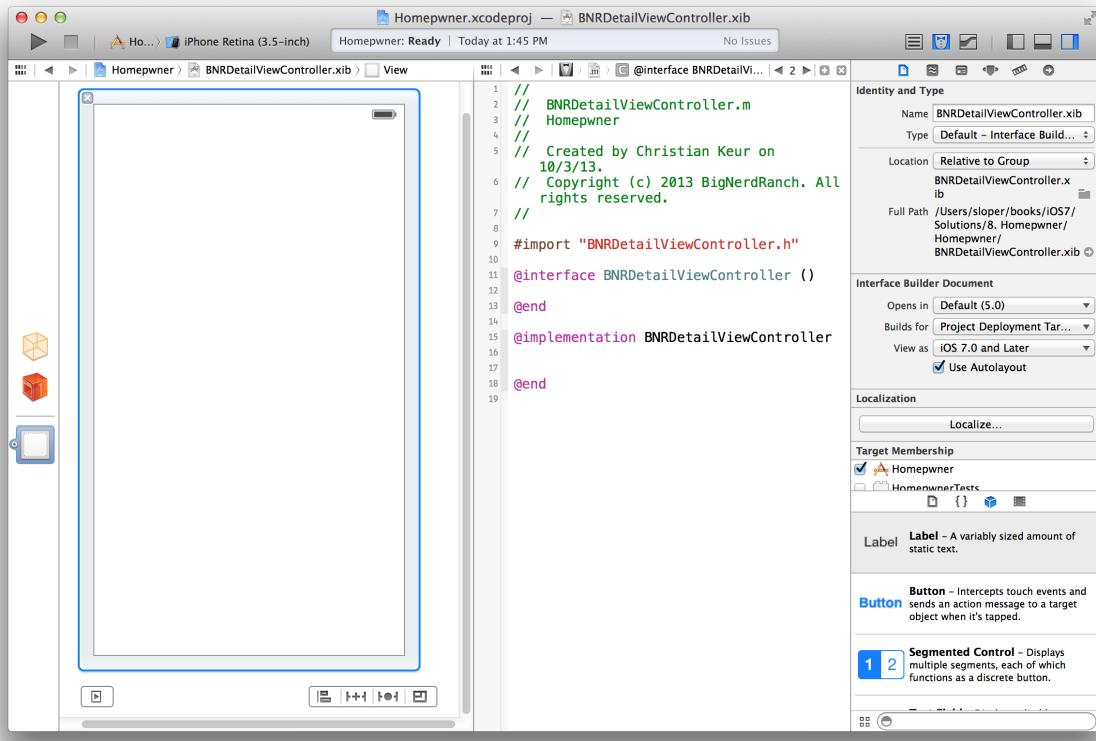
Nos exercícios anteriores, estas foram três etapas distintas: você adicionou outlets no arquivo da interface, depois configurou a interface no arquivo XIB e, em seguida, fez as conexões. Você pode combinar essas etapas usando um atalho no Xcode. Primeiro, abra o arquivo `BNRDetailViewController.xib` selecionando-o no navegador de projetos.

Agora, pressione Option e clique em `BNRDetailViewController.m` no navegador de projetos. Esse atalho abre o arquivo no *editor assistente*, bem ao lado de `BNRDetailViewController.xib`. (Você pode ativar/desativar o editor assistente clicando no botão do meio do controle Editor, na parte superior da área de trabalho. O atalho para exibir o editor assistente é Command-Option-Return. Para voltar ao editor padrão, use Command-Return.)

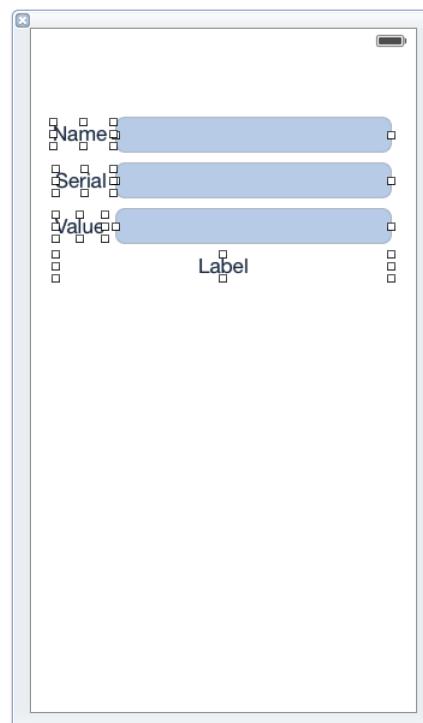
Você também precisará ter a biblioteca de objetos disponível para que possa arrastar as subvisões para a visão. Mostre a área de utilitários, clicando no botão direito no controle View, na parte superior da área de trabalho (ou Command-Option-0).

Sua janela agora está suficientemente desorganizada. Vamos abrir um pouco de espaço temporário. Esconda a área do navegador, clicando no botão esquerdo no controle View, na parte superior da área de trabalho (o atalho é Command-0). Em seguida, altere o dock no Interface Builder para mostrar a visão de ícone, clicando no botão de alternância localizado no canto inferior esquerdo do editor. Sua área de trabalho deve ficar semelhante à Figure 10.8.

Figure 10.8 Layout da área de trabalho



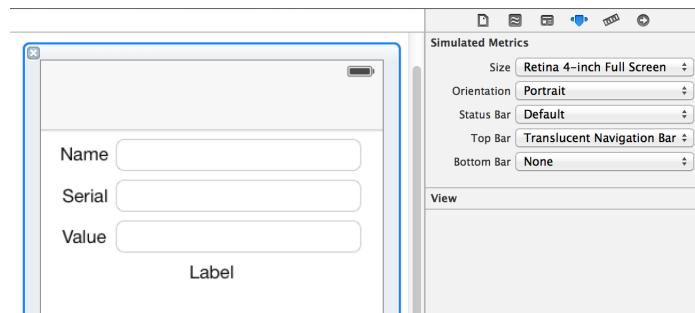
Agora, arraste quatro objetos de **UILabel** e três objetos de **UITextField** para a visão na área do canvas e configure-os para ficarem como na Figure 10.9.

Figure 10.9 XIB de **BNRDetailViewController** configurado

É importante que as subvisões que você acabou de adicionar não estejam posicionadas perto do topo do XIB. Isso se deve ao fato de que a visão de uma **UIViewController** se estende abaixo da **UINavigationBar** (o mesmo acontece com a **UITabBar**). Para facilitar a configuração das interfaces, a visão do nível raiz em um arquivo XIB possui *métricas simuladas* que mostrarão a você como a interface ficará com uma barra de navegação na parte superior. Você também pode visualizar uma barra de guias na parte inferior e uma série de outras situações em que a sua interface possa se encontrar.

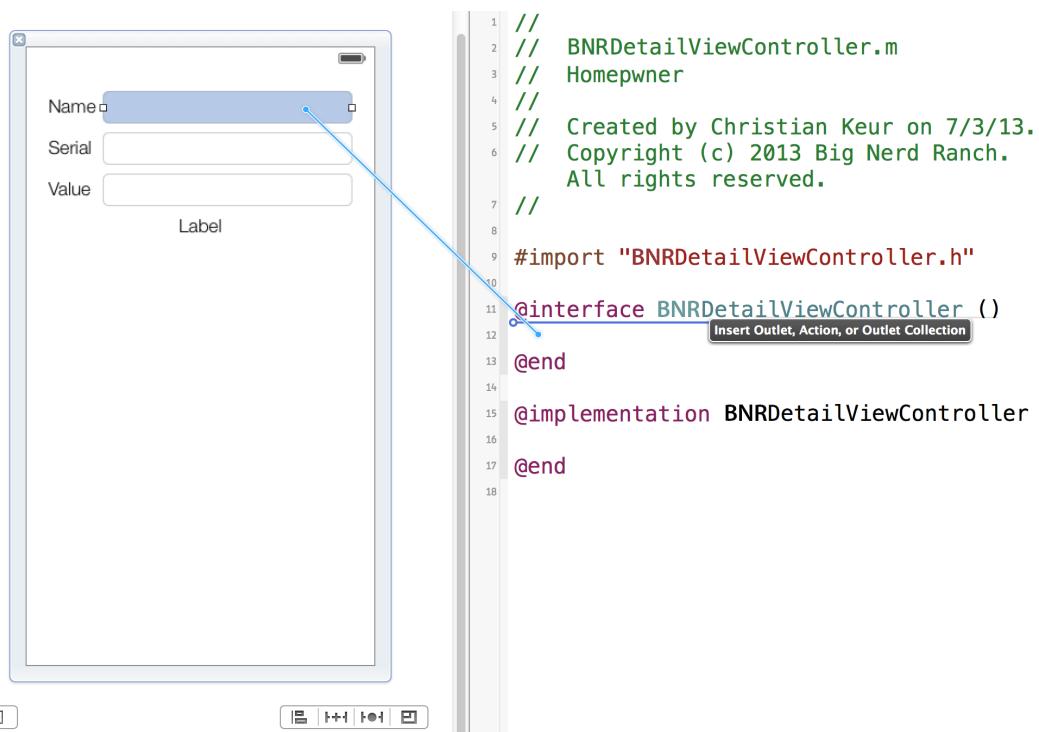
Para ver as métricas simuladas, selecione a visão do nível raiz e abra o inspetor de atributos. Na parte superior, você verá Simulated Metrics. Para Top Bar, selecione Translucent Navigation Bar (Figure 10.10).

Figure 10.10 Métricas simuladas



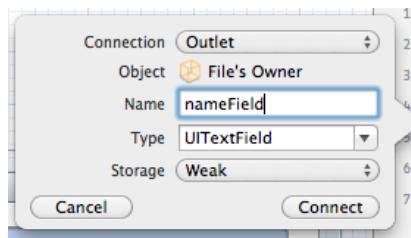
As três instâncias de **UITextField** e a instância inferior de **UILabel** serão os outlets em **BNRDetailViewController**. Aqui vem a parte estimulante. Pressionado a tecla Control, arraste da **UITextField**, ao lado do rótulo Name, para a extensão de classe no **BNRDetailViewController.m**, conforme mostrado na Figure 10.11.

Figure 10.11 Arrastando do arquivo XIB para o arquivo-fonte



Solte quando ainda estiver dentro da extensão de classe, e uma janela pop-up será exibida. Insira `nameField` no campo Name, selecione Weak no menu pop-up Storage e clique em Connect (Figure 10.12).

Figure 10.12 Geração automática de outlet e criação de uma conexão

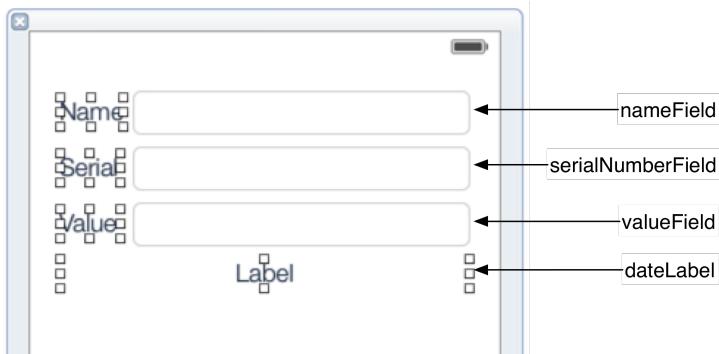


Isso criará uma propriedade `IBOutlet` do tipo `UITextField`, chamada `nameField`, em `BNRDetailViewController`. Você selecionou armazenamento `Weak` para essa propriedade porque o objeto para o qual ela apontará não é um objeto de alto nível no arquivo XIB.

Além disso, esse `UITextField` agora está conectado ao outlet `nameField` de File's Owner no arquivo XIB. Você pode confirmar isso, clicando no File's Owner com a tecla Control pressionada para ver as conexões. Observe também que ao passar o mouse sobre a conexão `nameField` no painel que aparece, o `UITextField` que você conectou será mostrado. Dois pássaros com uma pedra só.

Crie os outros três outlets da mesma maneira e coloque os nomes mostrados na Figure 10.13.

Figure 10.13 Diagrama de conexões



Após fazer as conexões, `BNRDetailViewController.m` deve ficar assim:

```
#import "BNRDetailViewController.h"

@interface BNRDetailViewController : UIViewController

@property (weak, nonatomic) IBOutlet UITextField *nameField;
@property (weak, nonatomic) IBOutlet UITextField *serialNumberField;
@property (weak, nonatomic) IBOutlet UITextField *valueField;
@property (weak, nonatomic) IBOutlet UILabel *dateLabel;

@end

@implementation BNRDetailViewController

@end
```

Se o seu arquivo estiver diferente, então os seus outlets não estão conectados direito.

Corrija qualquer diferença entre o seu arquivo e o código mostrado acima em três etapas: Primeiro, realize o processo de arrastar com a tecla Control pressionada e faça as conexões novamente, até que você tenha as quatro linhas mostradas acima em seu `BNRDetailViewController.m`. Segundo, retire qualquer código errado (como declarações de métodos de não propriedade ou variáveis de instância) que tenha sido criado. Por fim, verifique se há qualquer conexão ruim no arquivo XIB. No `BNRDetailViewController.xib`, pressione a tecla Control e

clique no File's Owner. Caso haja sinais de aviso em amarelo ao lado de alguma conexão, clique no ícone x ao lado destas conexões para desconectá-las.

É importante garantir que não haja nenhuma conexão ruim no arquivo XIB. Uma conexão ruim normalmente acontece quando você muda o nome de uma variável de instância, mas não atualiza a conexão no arquivo XIB. Ou, você remove uma variável de instância completamente, mas não a remove do arquivo XIB. De qualquer maneira, uma conexão ruim fará com que o aplicativo falhe quando o arquivo XIB for carregado.

Agora, vamos fazer mais conexões. Para cada instância de **UITextField** no arquivo XIB, conecte a propriedade delegate ao File's Owner. (Lembre-se, pressione Control e arraste de **UITextField** para o File's Owner e selecione delegate na lista.)

Agora que este projeto tem um bom número de arquivos-fonte, você estará mudando de um para outro com bastante regularidade. Uma forma de acelerar esta alternância entre arquivos comumente acessados é usar as guias do Xcode. Se clicar duas vezes em um arquivo no navegador de projetos, o arquivo abrirá em uma nova guia. Você também pode abrir uma guia em branco com o atalho Command-T. Os atalhos de teclado para alternar entre guias são Command-Shift-} e Command-Shift-{. (Você pode ver outros atalhos para a organização do projeto selecionando a guia General das preferências do Xcode.)

## Navegação com UINavigationController

Agora, você tem um controlador de navegação e duas subclasses de controlador de visão. Está na hora da montagem final. O usuário deve ser capaz de tocar em uma linha na visão de tabela do **BNRItemsViewController** e fazer com que a visão do **BNRDetailViewController** deslize para a tela e as propriedades da instância de **BNRItem** selecionada sejam exibidas.

### Envio de controladores de visão

Claro, você precisa criar uma instância de **BNRDetailViewController**. Onde este objeto deve ser criado? Pense nos exercícios anteriores, em que você criou instâncias de todos os controladores do método **application:didFinishLaunchingWithOptions:**. Por exemplo, no Chapter 6, você criou ambos os controladores de visão e os adicionou imediatamente ao array **viewControllers** do controlador da barra de guias.

No entanto, ao usar uma **UINavigationController**, você não pode simplesmente armazenar todos os controladores de visão possíveis na pilha. O array **viewControllers** de um controlador de navegação é dinâmico – você começa com um controlador de visão raiz e envia controladores de visão, dependendo da entrada de dados do usuário. Portanto, algum objeto, que não seja o controlador de navegação, precisa criar a instância de **BNRDetailViewController** e ser responsável por adicioná-la à pilha.

Esse objeto deve atender a dois requisitos: tem que saber quando enviar **BNRDetailViewController** para a pilha, e ela precisa de um ponteiro para o controlador de navegação de modo a enviar mensagens a ele, a saber, **pushViewController:animated:**.

**BNRItemsViewController** atende aos dois requisitos. Primeiro, ele sabe quando uma linha é tocada em uma visão de tabela porque, como seu delegate, ele recebe a mensagem **tableView:didSelectRowAtIndexPath:** quando este evento ocorre. Segundo, qualquer controlador de visão em uma pilha do controlador de navegação pode colocar um ponteiro para este controlador de navegação, enviando a si mesmo a mensagem **navigationController**. Como o controlador de visão raiz, **BNRItemsViewController** está sempre na pilha do controlador de navegação e, por isso, pode sempre acessá-lo.

Portanto, **BNRItemsViewController** será responsável pela criação da instância de **BNRDetailViewController** e por adicioná-la à pilha. Na parte superior do **BNRItemsViewController.h**, importe o arquivo de cabeçalho para **BNRDetailViewController**.

```
#import "BNRDetailViewController.h"

@interface BNRItemsViewController : UITableViewController
```

Quando uma linha recebe um toque na visão de tabela, seu delegate recebe a mensagem **tableView:didSelectRowAtIndexPath:**, que contém o caminho do índice da linha selecionada. No

`BNRItemsViewController.m`, implemente esse método para criar uma `BNRDetailViewController` e, em seguida, envie-a para o topo da pilha do controlador de navegação.

```
@implementation BNRItemsViewController
- (void)tableView:(UITableView *)tableView
didSelectRowAtIndexPath:(NSIndexPath *)indexPath
{
    BNRDetailViewController *detailViewController =
        [[BNRDetailViewController alloc] init];
    // Push it onto the top of the navigation controller's stack
    [self.navigationController pushViewController:detailViewController
                                    animated:YES];
}
```

Compile e execute o aplicativo. Crie um novo item e selecione essa linha da `UITableView`. Você não só é direcionado para a view da `BNRDetailViewController`, como também recebe uma animação livre e um botão de voltar na `UINavigationBar`. Toque neste botão para voltar para `BNRItemsViewController`.

Uma vez que a pilha de `UINavigationController` é um array, ela será a proprietária de qualquer controlador de visão adicionado a ela. Deste modo, `BNRDetailViewController` é propriedade apenas de `UINavigationController` depois que o método `tableView:didSelectRowAtIndexPath:` termina. Quando a pilha é disparada, a `BNRDetailViewController` é destruída. A próxima vez que uma linha é tocada, uma nova instância de `BNRDetailViewController` é criada.

Um controlador de visão enviar o próximo controlador de visão é um padrão comum. O controlador de visão raiz normalmente cria o próximo controlador de visão, o próximo controlador de visão cria o seguinte, e assim por diante. Alguns aplicativos podem ter controladores de visão que conseguem enviar controladores de visão diferentes, dependendo da entrada de dados do usuário. Por exemplo, o aplicativo Photos envia um controlador de visão de vídeo ou um controlador de visão de imagem para a pilha de navegação, dependendo do tipo de mídia selecionado.

(A classe `UISplitViewController` somente para iPad requer um padrão diferente. A tela maior do iPad permite que dois controladores de visão em uma interface drill-down apareçam na tela simultaneamente, em vez de serem enviados para a mesma pilha. Você saberá mais sobre `UISplitViewController` no Chapter 22.)

## Passando dados entre controladores de visão

Claro, os campos de texto na tela estão vazios no momento. Para preencher estes campos, você precisa de uma maneira de passar a classe `BNRItem` selecionada de `BNRItemsViewController` para `BNRDetailViewController`.

Para conseguir fazer isso, você dará à `BNRDetailViewController` uma propriedade para conter uma classe `BNRItem`. Quando uma linha é tocada, `BNRItemsViewController` dará a `BNRItem` correspondente para a instância de `BNRDetailViewController`, que está sendo enviada para a pilha. A `BNRDetailViewController` preencherá os campos de texto com as propriedades dessa `BNRItem`. Editar o texto nos campos de texto na view de `BNRDetailViewController` alterará as propriedades dessa `BNRItem`.

No `BNRDetailViewController.h`, adicione essa propriedade. Além disso, na parte superior deste arquivo, faça a declaração forward de `BNRItem`.

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

@class BNRItem;

@interface BNRDetailViewController : UIViewController
@property (nonatomic, strong) BNRItem *item;
@end
```

No `BNRDetailViewController.m`, importe o arquivo de cabeçalho de `BNRItem`.

```
#import "BNRItem.h"
```

Quando a view de **BNRDetailViewController** é exibida na tela, ela precisa configurar suas subvisões para mostrar as propriedades de item. No **BNRDetailViewController.m**, sobrescreva **viewWillAppear:** para transferir as propriedades de item às várias instâncias de **UITextField**.

```
- (void)viewWillAppear:(BOOL)animated
{
    [super viewWillAppear:animated];

    BNRItem *item = self.item;

    self.nameField.text = item.itemName;
    self.serialNumberField.text = item.serialNumber;
    self.valueField.text = [NSString stringWithFormat:@"%@", item.valueInDollars];

    // You need an NSDateFormatter that will turn a date into a simple date string
    static NSDateFormatter *dateFormatter = nil;
    if (!dateFormatter) {
        dateFormatter = [[NSDateFormatter alloc] init];
        dateFormatter.dateStyle = NSDateFormatterMediumStyle;
        dateFormatter.timeStyle = NSDateFormatterNoStyle;
    }

    // Use filtered NSDate object to set dateLabel contents
    self.dateLabel.text = [dateFormatter stringFromDate:item.dateCreated];
}
```

No **BNRItemsViewController.m**, acrescente o código a seguir a **tableView:didSelectRowAtIndexPath:** para que **BNRDetailViewController** tenha sua variável item antes que o método **viewWillAppear:** seja chamado.

```
- (void)tableView:(UITableView *)tableView
    didSelectRowAtIndexPath:(NSIndexPath *)indexPath
{
    BNRDetailViewController *detailViewController =
        [[BNRDetailViewController alloc] init];

    NSArray *items = [[BNRItemStore sharedStore] allItems];
    BNRItem *selectedItem = items[indexPath.row];

    // Give detail view controller a pointer to the item object in row
    detailViewController.item = selectedItem;

    [self.navigationController pushViewController:detailViewController
        animated:YES];
}
```

Muitos programadores iniciantes em iOS têm dificuldade com a forma como dados são passados entre controladores de visão. Ter todos os dados no controlador de visão raiz e passar subconjuntos de dados para o próximo **UIViewController** (como você acabou de fazer) é uma forma limpa e eficiente de realizar essa tarefa.

Compile e execute seu aplicativo. Crie um novo item e selecione essa linha na **UITableView**. A visão que aparece conterá as informações para a **BNRItem** selecionada. Embora você possa editar esses dados, a **UITableView** não refletirá as alterações quando você retornar a ela. Para corrigir este problema, você precisa implementar o código para atualizar as propriedades do **BNRItem** que está sendo editado. Na próxima seção, veremos quando fazer isso.

## Fazendo com que visões apareçam e desapareçam

Sempre que uma classe **UINavigationController** está prestes a alternar visões, ela envia duas mensagens: **viewWillDisappear:** e **viewWillAppear:**. O **UIViewController** que está prestes a ser retirado da pilha recebe a mensagem **viewWillDisappear:**. O **UIViewController** que depois ficará no início da pilha recebe **viewWillAppear:**.

Quando uma **BNRDetailViewController** for retirada da pilha, você definirá as propriedades de sua item para o conteúdo dos campos de texto. Ao implementar estes métodos para as visões que aparecem e desaparecem,

é importante chamar a implementação da superclasse – ela pode ter trabalho a fazer e precisa receber a oportunidade para fazê-lo. No `BNRDetailViewController.m`, implemente `viewWillDisappear:`:

```
- (void)viewWillDisappear:(BOOL)animated
{
    [super viewWillDisappear:animated];

    // Clear first responder
    [self.view endEditing:YES];

    // "Save" changes to item
    BNRItem *item = self.item;
    item.itemName = self.nameField.text;
    item.serialNumber = self.serialNumberField.text;
    item.valueInDollars = [self.valueField.text intValue];
}
```

Observe o uso de `endEditing:`. Quando a mensagem `endEditing:` é enviada para uma visão, se ela ou alguma de suas subvisões for, no momento, o primeiro respondente, ela renunciará ao status de primeiro respondente e o teclado será fechado. (O argumento passado determina se o primeiro respondente deve ser forçado a se retirar. Alguns primeiros respondentes podem se recusar a renunciar e passar YES ignora essa recusa.)

Agora, os valores de `BNRItem` serão atualizados quando o usuário tocar no botão Back na `UINavigationBar`. Quando `BNRItemsViewController` é exibido novamente na tela, recebe a mensagem `viewWillAppear:`.

Aproveite essa oportunidade para recarregar a `UITableView`, para que o usuário possa ver as alterações imediatamente. No `BNRItemsViewController.m`, sobrescreva o `viewWillAppear:`.

```
- (void)viewWillAppear:(BOOL)animated
{
    [super viewWillAppear:animated];
    [self.tableView reloadData];
}
```

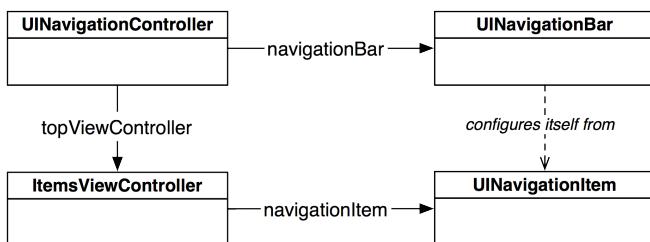
Compile e execute seu aplicativo. Agora você pode passar entre os controladores de visão que criou e alterar os dados com facilidade.

## UINavigationBar

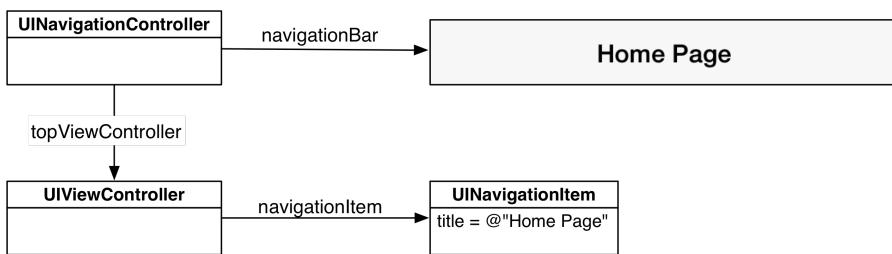
A `UINavigationBar` não é muito interessante neste momento. Uma `UINavigationBar` deve exibir um título descritivo para o `UIViewController` que está atualmente no topo da pilha do `UINavigationController`.

Todo `UIViewController` tem uma propriedade `navigationItem` do tipo `UINavigationItem`. No entanto, ao contrário da `UINavigationBar`, o `UINavigationItem` não é uma subclasse de `UIView`, assim, não pode aparecer na tela. Em vez disso, o item de navegação fornece à barra de navegação o conteúdo que precisa extrair. Quando uma `UIViewController` chega ao topo da pilha de `UINavigationController`, a `UINavigationBar` usa a `navigationItem` de `UIViewController` para configurar a si mesma, como mostrado na Figure 10.14.

Figure 10.14 `UINavigationItem`



Por padrão, uma `UINavigationItem` é vazia. No nível mais básico, uma `UINavigationItem` tem uma string simples de `title` (título). Quando uma `UIViewController` é movida para o topo da pilha de navegação e sua `navigationItem` tem uma string válida para a propriedade `title`, a barra de navegação exibirá essa string (Figure 10.15).

Figure 10.15 **UINavigationItem** com título

No **BNRItemsViewController.m**, modifique **init** para definir a **title** de **navigationItem** como **Homepwner**.

```

- (instancetype)init
{
    self = [super initWithStyle:UITableViewStylePlain];
    if (self) {
        UINavigationItem *navItem = self.navigationItem;
        navItem.title = @"Homepwner";
    }
    return self;
}
  
```

Compile e execute o aplicativo. Observe a string **Homepwner** na barra de navegação. Crie e toque em uma linha e observe que a barra de navegação não tem mais um título. Você precisa dar um título à **BNRDetailViewController**, também. Seria interessante se o título do item de navegação do **BNRDetailViewController** fosse o nome do **BNRItem** que ele está exibindo. Obviamente, não é possível fazer isso em **init**, pois você ainda não sabe qual será sua item.

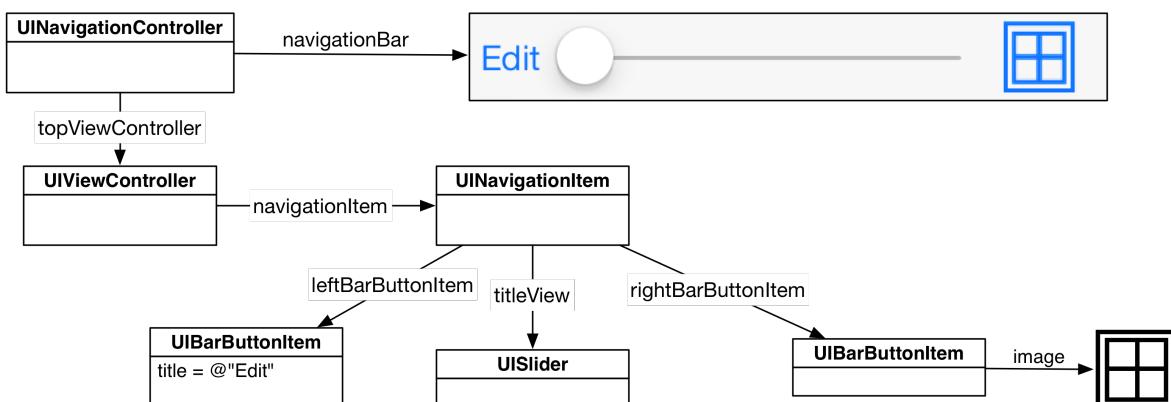
Em vez disso, a **BNRDetailViewController** definirá seu título quando definir a propriedade **item**. No **BNRDetailViewController.m**, implemente **setItem:**, substituindo o método setter sintetizado para **item**.

```

- (void)setItem:(BNRItem *)item
{
    _item = item;
    self.navigationItem.title = _item.itemName;
}
  
```

Compile e execute o aplicativo. Crie e toque em uma linha e você verá que o título da barra de navegação é o nome da **BNRItem** que você selecionou.

Um item de navegação pode conter mais do que apenas uma string de título, como mostrado na Figure 10.16. Existem três áreas passíveis de customização para cada **UINavigationItem**: **leftBarButtonItem**, **rightBarButtonItem** e **titleView**. Os itens de botão da barra da esquerda e da direita são ponteiros para instâncias de **UIBarButtonItem**, que contém as informações para um botão que pode ser exibido apenas em uma **UINavigationBar** ou uma **UIToolbar**.

Figure 10.16 **UINavigationItem** completa

Assim como **UINavigationItem**, **UIBarButtonItem** não é uma subclasse de **UIView**. Em vez disso, **UINavigationItem** condensa as informações que a **UINavigationBar** utiliza para configurar a si mesma. Da mesma forma, **UIBarButtonItem** não é uma visão, mas contém informações sobre como um único botão na **UINavigationBar** deve ser exibido. (A **UIToolbar** também usa instâncias de **UIBarButtonItem** para configurar a si mesma.)

A terceira área passível de customização de uma **UINavigationItem** é a sua **titleView**. Você pode usar uma string básica como título ou fazer com que uma subclasse da **UIView** fique no centro do item de navegação. Não é possível ter ambos. Se for adequado para o contexto de um controlador de visão específico ter uma visão personalizada (como um botão, uma barra deslizante, uma imagem ou até mesmo um mapa), você definiria a **titleView** do item de navegação para essa visão personalizada. A Figure 10.16 mostra um exemplo de uma **UINavigationItem** com uma visão personalizada como sua **titleView**. Contudo, normalmente uma string de título é suficiente, e é isso que você fará neste capítulo.

Vamos adicionar uma **UIBarButtonItem** à **UINavigationBar**. Você quer que este botão fique à direita da barra de navegação quando **BNRItemsViewController** estiver no início da pilha. Quando tocado, ele deve adicionar uma nova **BNRItem** à lista.

Um item de botão de barra tem um par destino-ação que funciona como o mecanismo de destino-ação de **UIControl**: quando tocado, ele envia uma mensagem de ação ao destino. Ao definir um par destino-ação em um arquivo XIB, você pressiona Control e arrasta de um botão ao seu destino e, em seguida, seleciona um método da lista de **IBActions**. Para configurar um par destino-ação de forma programática, você passa o destino e a ação ao botão.

No **BNRItemsViewController.m**, crie uma instância de **UIBarButtonItem** e dê a ela seu destino e ação.

```
- (instancetype)init
{
    self = [super initWithStyle:UITableViewStylePlain];
    if (self) {
        UINavigationItem *navItem = self.navigationItem;
        navItem.title = @"Homepwner";

        // Create a new bar button item that will send
        // addNewItem: to BNRItemsViewController
        UIBarButtonItem *bbi = [[UIBarButtonItem alloc]
                               initWithBarButtonSystemItem:UIBarButtonSystemItemAdd
                               target:self
                               action:@selector(addNewItem:)];
        // Set this bar button item as the right item in the navigationItem
        navItem.rightBarButtonItem = bbi;
    }
    return self;
}
```

A ação é passada como um valor do tipo SEL. Lembre-se que o tipo de dados SEL é um ponteiro para um seletor e que um seletor é o nome inteiro da mensagem, incluindo quaisquer dois pontos. Observe que `@selector()` não se importa com o tipo de retorno, tipos ou nomes de argumentos.

Compile e execute o aplicativo. Toque no botão + e uma nova linha aparecerá na tabela. (Observe que esta não é a única maneira de configurar um item de botão de barra. Consulte a documentação para conhecer outras mensagens de inicialização que você pode enviar para uma instância de **UIBarButtonItem**.)

Agora, vamos adicionar outra **UIBarButtonItem** para substituir o botão Edit no cabeçalho da visão de tabela. No **BNRItemsViewController.m**, edite o método **init**.

```
- (instancetype)init
{
    self = [super initWithStyle:UITableViewStylePlain];
    if (self) {
        UINavigationItem *navItem = self.navigationItem;
        navItem.title = @"Homepwner";

        // Create a new bar button item that will send
        // addNewItem: to BNRIItemsViewController
        UIBarButtonItem *bbi = [[UIBarButtonItem alloc]
            initWithFrame:UIBarButtonItemSystemItemAdd
            target:self
            action:@selector(addNewItem:)];

        // Set this bar button item as the right item in the navigationItem
        navItem.rightBarButtonItem = bbi;

        navItem.leftBarButtonItem = self.editButtonItem;
    }
    return self;
}
```

Surpreendentemente, este é todo o código que você precisa para obter um botão de edição na barra de navegação. Compile e execute, toque no botão Edit e veja a **UITableView** entrar no modo de edição! De onde o **editButtonItem** vem? **UIViewController** tem uma propriedade **editButtonItem** e, quando recebe **editButtonItem**, o controlador de visão cria uma **UIBarButtonItem** com o título Edit. Melhor ainda, este botão vem com um par destino-ação: ele envia a mensagem **setEditing:animated:** para seu **UIViewController** quando tocado.

Agora que Homepwner tem uma barra de navegação totalmente funcional, você pode se livrar da visão de cabeçalho e do código relacionado. No **BNRIItemsViewController.m**, exclua os seguintes métodos:

```
-(UIView *)tableView:(UITableView *)tv
    viewForHeaderInSection:(NSInteger)section
{
    return self.headerView;
}

-(CGFloat)tableView:(UITableView *)tv
    heightForHeaderInSection:(NSInteger)section
{
    return self.headerView.frame.size.height;
}

-(UIView *)headerView
{
    if (!headerView) {
        [[NSBundle mainBundle] loadNibNamed:@"HeaderView" owner:self options:nil];
    }
    return _headerView;
}

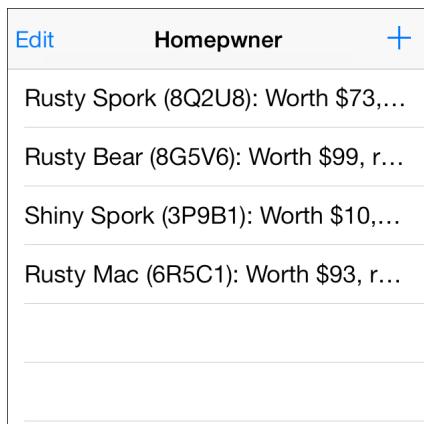
-(IBAction)toggleEditingStyle:(id)sender
{
    if (self.isEditing) {
        [sender setTitle:@"Edit" forState:UIControlStateNormal];
        [self setEditing:NO animated:YES];
    } else {
        [sender setTitle:@"Done" forState:UIControlStateNormal];
        [self setEditing:YES animated:YES];
    }
}
```

Você pode também excluir a declaração da propriedade **headerView**.

Finalmente, você também pode remover o arquivo **HeaderView.xib** do navegador de projetos.

Compile e execute novamente. Os antigos botões Edit e New desapareceram, deixando você com uma bela **UINavigationBar** (Figure 10.17).

Figure 10.17 Homepwner com barra de navegação



## Desafio de bronze: exibir um teclado numérico

O teclado para a classe `UITextField` que exibe uma `valueInDollars` de `BNRItem` é um teclado QWERTY. Seria melhor se fosse um teclado numérico. Altere o Keyboard Type deste `UITextField` para Number Pad. (Dica: você pode fazer isso no arquivo XIB, usando o inspetor de atributos.)

## Desafio de prata: fechar um teclado numérico

Depois de completar o desafio de bronze, você pode perceber que não há nenhuma tecla de retorno no teclado numérico. Crie uma maneira de o usuário tirar o teclado numérico da tela.

## Desafio de ouro: enviar mais controladores de visão

Neste momento, as instâncias de `BNRItem` não podem ter sua propriedade `dateCreated` alterada. Altere a `BNRItem` para que elas possam ter sua propriedade alterada e, em seguida, adicione um botão abaixo de `dateLabel` em `BNRDetailViewController` com o título `Change Date` (alterar data). Quando esse botão for tocado, envie outra instância de controlador de visão para a pilha de navegação. Esse controlador de visão deve ter uma instância de `UIDatePicker` que modifica a propriedade `dateCreated` da `BNRItem` selecionada.

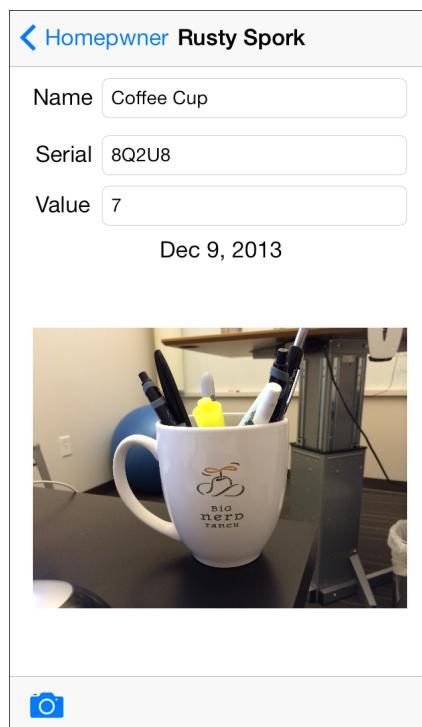


# 11

## Câmera

Neste capítulo, você vai adicionar fotos ao aplicativo Homepwner. Você apresentará uma **UIImagePickerController** para que o usuário possa tirar e salvar uma foto de cada item. Depois, a imagem será associada a uma instância de **BNRItem** e ficará visível na visão de detalhes do item.

Figure 11.1 Homepwner com adição de câmera



As imagens costumam ser bastante grandes, portanto, recomenda-se armazenar as imagens separadamente de outros dados. Sendo assim, neste capítulo, você criará um segundo armazenamento para imagens. A **BNRImageStore** buscará as imagens e as armazenará em cache conforme necessário. Você também poderá esvaziar o cache quando a memória ficar com pouco espaço.

### Exibição de imagens e UIImageView

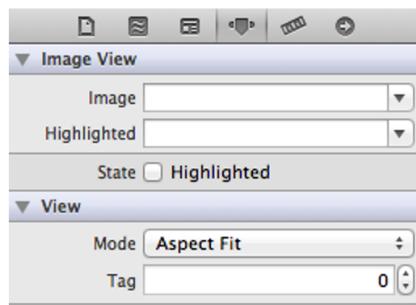
A primeira etapa é fazer a **BNRDetailViewController** obter e exibir uma imagem. Uma maneira fácil de exibir uma imagem é colocar uma instância de **UIImageView** na tela.

Abra o `Homepwner.xcodeproj` e o `BNRDetailViewController.xib`. Em seguida, arraste uma instância de **UIImageView** para a view e posicione-a abaixo do rótulo. Redimensione a visão da imagem de forma que fique quase com a mesma largura da tela, mas deixe algum espaço na parte inferior para uma possível barra de ferramentas (Figure 11.2).

Figure 11.2 **UIImageView** na visão de **BNRDetailViewController**

Uma **UIImageView** exibe uma imagem de acordo com a sua propriedade `contentMode`. Essa propriedade determina onde posicionar e como redimensionar o conteúdo dentro do frame da visão de imagem. O valor padrão de **UIImageView** para `contentMode` é `UIViewContentModeScaleToFill`, que ajustará a imagem para coincidir exatamente com os limites da visão de imagem. Caso mantenha o padrão, a imagem capturada pela câmera será distorcida para caber no quadrado da **UIImageView**. Você precisa mudar a `contentMode` da visão de imagem para que ela redimensione a imagem com a mesma proporção de tela.

Selecione **UIImageView** e abra o inspetor de atributos. Encontre o atributo `Mode` e mude-o para `Aspect Fit` (Figure 11.3). Isso redimensionará a imagem para caber dentro dos limites de **UIImageView**.

Figure 11.3 Alteração do modo da **UIImageView** para ajustar aspecto

Agora, pressione a tecla Option e clique em `BNRDetailViewController.m`, no navegador de projetos, para abri-lo no editor assistente. Pressionando a tecla Control, arraste-o da **UIImageView** para a extensão de classe no `BNRDetailViewController.m`. Dê o nome `imageView` para o outlet e escolha `Weak` como o tipo de armazenamento. Clique em Connect.

A extensão de classe da **BNRDetailViewController** agora deve ser semelhante ao seguinte:

```

@interface BNRDetailViewController : UIViewController

@property (weak, nonatomic) IBOutlet UITextField *nameField;
@property (weak, nonatomic) IBOutlet UITextField *serialNumberField;
@property (weak, nonatomic) IBOutlet UITextField *valueField;
@property (weak, nonatomic) IBOutlet UILabel *dateLabel;
@property (weak, nonatomic) IBOutlet UIImageView *imageView;

@end

```

## Adição de um botão de câmera

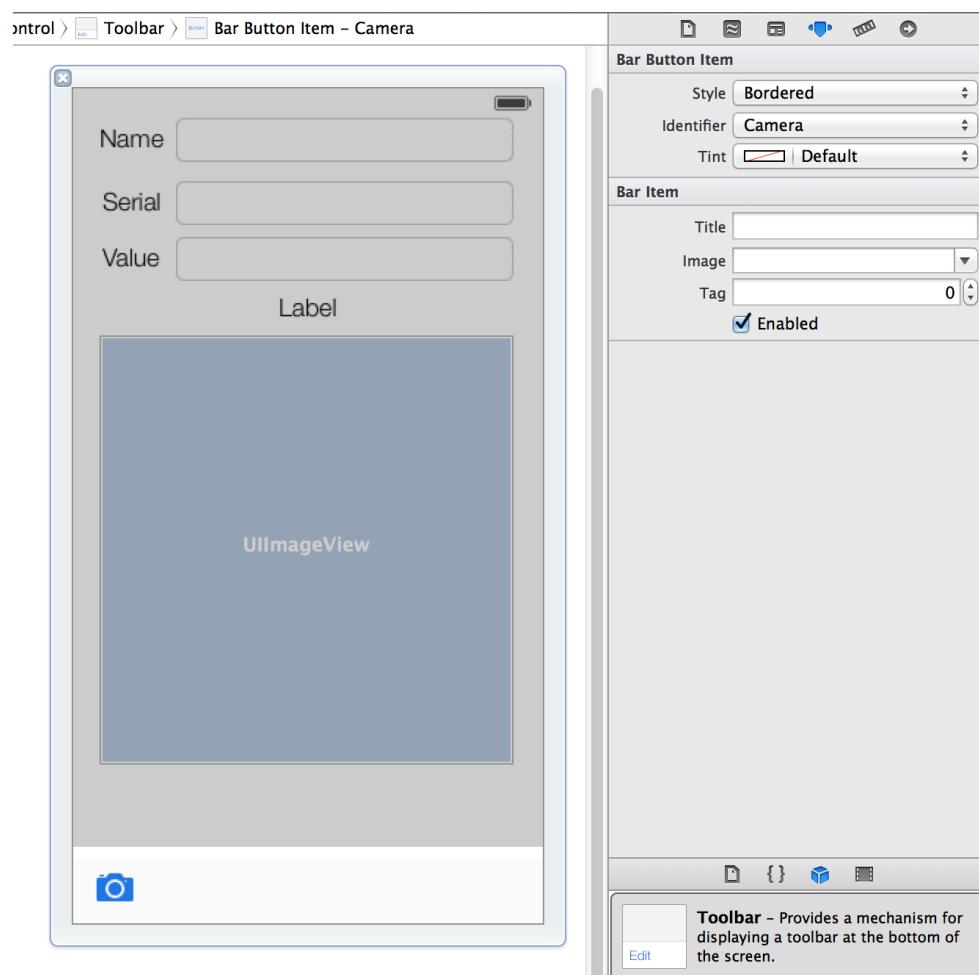
Agora, você precisa de um botão para iniciar o processo de tirar fotos. Seria interessante colocar esse botão na barra de navegação, mas você precisará da barra de navegação para outro botão mais tarde. Em vez disso, você criará uma instância de **UIToolbar** e a colocará na parte inferior da view de **BNRDetailViewController**.

No **BNRDetailViewController.xib**, arraste uma **UIToolbar** da biblioteca de objetos para a parte inferior da view.

Uma **UIToolbar** funciona muito como uma **UINavigationBar** – é possível adicionar instâncias de **UIBarButtonItem** a ela. No entanto, quando a barra de navegação tem dois slots para itens de botão de barra, uma barra de ferramentas tem um array para itens de botão de barra. Você pode posicionar quantos itens de botão de barra quiser em uma barra de ferramentas, desde que caibam na tela.

Por padrão, uma nova instância de **UIToolbar**, criada em um arquivo XIB, vem com uma **UIBarButtonItem**. Selecione este item de botão de barra e abra o inspetor de atributos. Mude o Identifier para Camera; o item mostrará um ícone de câmera (Figure 11.4).

Figure 11.4 **UIToolbar** com item de botão de barra

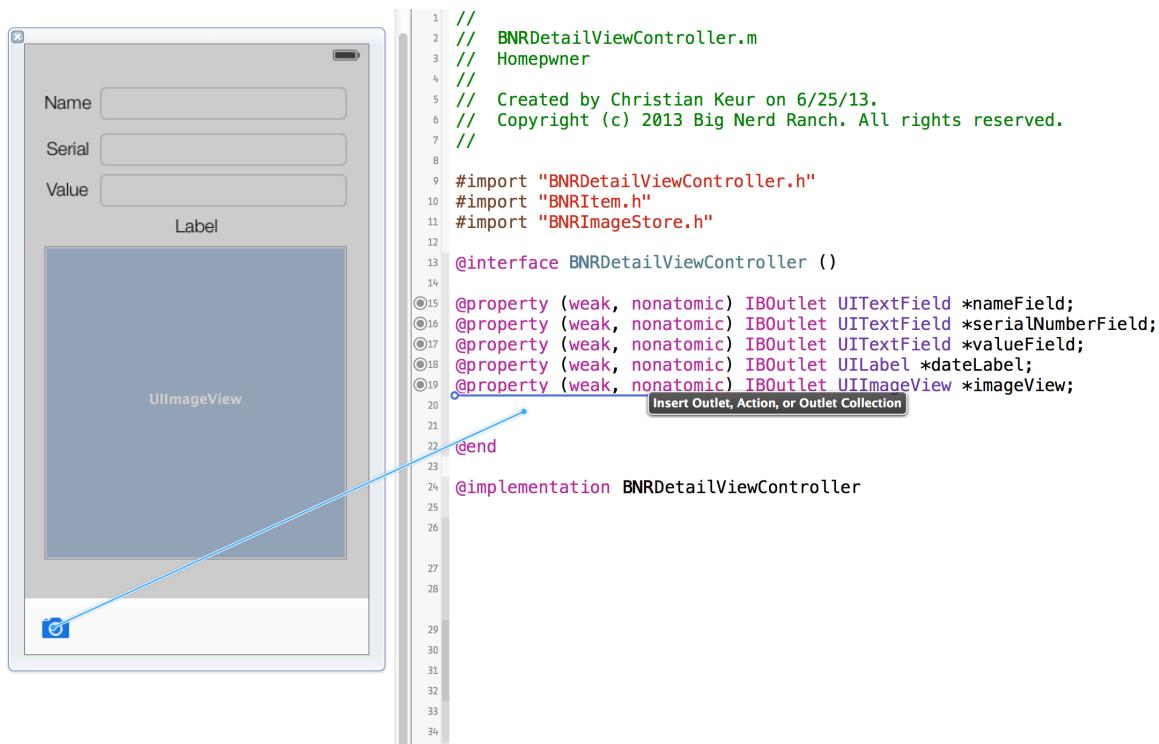


O botão da câmera precisa de um destino e uma ação. Em exercícios anteriores, você fez a conexão de um método de ação em duas etapas: declarando-o no código e fazendo a conexão no arquivo XIB. Assim como com os outlets, existe uma maneira de realizar as duas etapas de uma vez.

No navegador de projetos, pressione a tecla Option e clique em `BNRDetailViewController.m` para abri-lo no editor assistente.

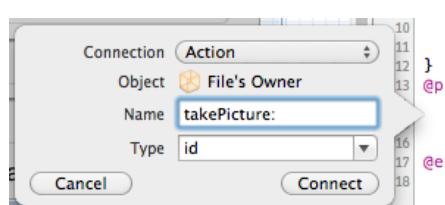
No `BNRDetailViewController.xib`, selecione o botão da câmera clicando primeiro na barra de ferramentas e depois no próprio botão. Em seguida, pressione Control e arraste o botão selecionado para a parte de implementação de `BNRDetailViewController.m` (Figure 11.5).

Figure 11.5 Criação e conexão de um método de ação a partir de um XIB



Solte o mouse; uma janela aparecerá e permitirá que você especifique o tipo de conexão que está criando. No menu pop-up Connection, selecione Action. Depois, coloque o nome `takePicture:` nesse método e clique em Connect (Figure 11.6).

Figure 11.6 Criação da ação

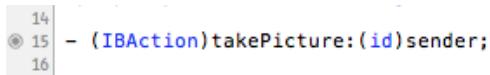


Agora, o stub do método de ação está no `BNRDetailViewController.m`, e a instância de `UIBarButtonItem` no XIB é conectada para enviar essa mensagem para a `BNRDetailViewController` quando tocada. O stub deverá ter a seguinte aparência:

```
- (IBAction)takePicture:(id)sender
{
}
```

O Xcode é inteligente o suficiente para saber quando um método de ação está conectado no arquivo XIB. Está vendo aquele pequeno círculo dentro de um círculo, na área da medianiz, ao lado do método **takePicture:** (Figure 11.7)? Quando esse círculo está preenchido, esse método de ação está conectado a um arquivo XIB. Um círculo vazio indica que ainda precisa ser conectado.

Figure 11.7 Status da conexão do arquivo-fonte



Em um capítulo posterior, você precisará de um ponteiro para a **UIToolbar** em si. Vamos fazer essa configuração agora. Selecione a barra de ferramentas (não o botão Camera na barra de ferramentas). Em seguida, pressione Control e arraste-a para a extensão de classe no **BNRDetailViewController.m**. Nomeie a toolbar desse outlet e certifique-se de que seu armazenamento seja Weak.

A interface para **BNRDetailViewController** agora possui um outlet da toolbar:

```
@interface BNRDetailViewController : UIViewController

@property (weak, nonatomic) IBOutlet UITextField *nameField;
@property (weak, nonatomic) IBOutlet UITextField *serialNumberField;
@property (weak, nonatomic) IBOutlet UITextField *valueField;
@property (weak, nonatomic) IBOutlet UILabel *dateLabel;
@property (weak, nonatomic) IBOutlet UIImageView *imageView;
@property (weak, nonatomic) IBOutlet UIToolbar *toolbar;

@end
```

Se você cometeu algum erro durante essas conexões, será necessário abrir o **BNRDetailViewController.xib** e desconectar conexões deficientes. (Procure por sinais de aviso em amarelo no inspetor de conexões.)

## Captura de fotos e UIImagePickerController

No método **takePicture:**, você irá criar uma instância de **UIImagePickerController** e apresentá-la na tela. Ao criar uma instância de **UIImagePickerController**, você deve definir a propriedade **sourceType** e atribuir um delegate a ela.

### Configuração de **sourceType** do seletor de imagens

A **sourceType** é uma constante que informa o seletor de imagens acerca de onde deve obter imagens. Ela tem três valores possíveis:

**UIImagePickerControllerSourceTypeCamera**

O usuário vai tirar uma nova foto.

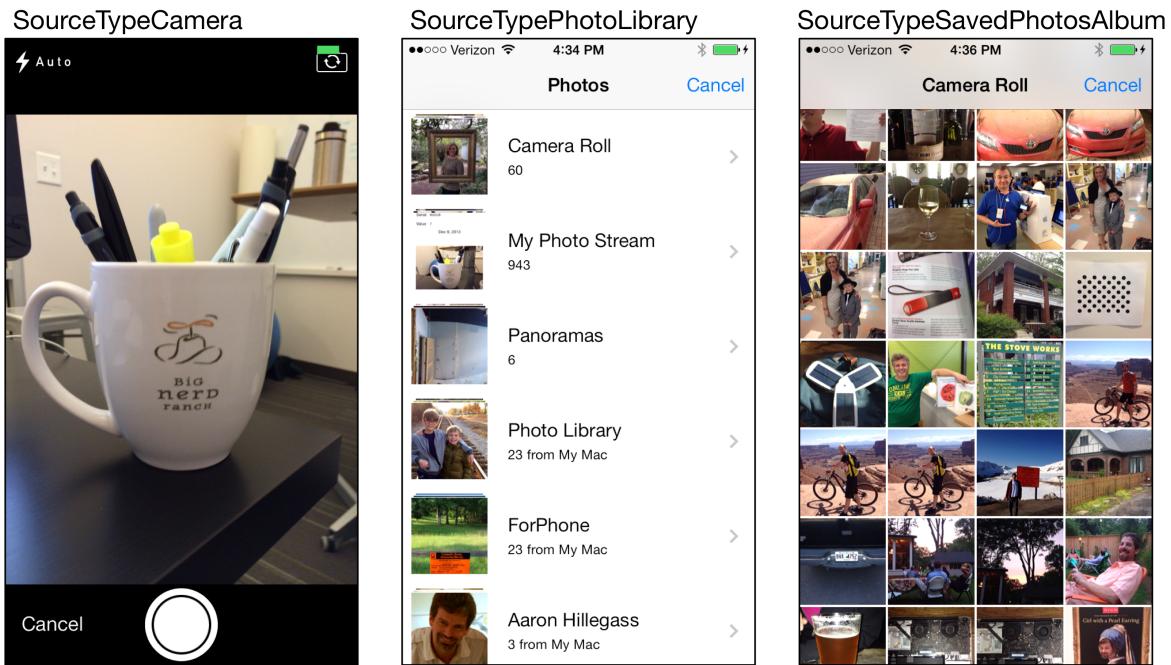
**UIImagePickerControllerSourceTypePhotoLibrary**

O usuário será solicitado a selecionar um álbum e, em seguida, uma foto desse álbum.

**UIImagePickerControllerSourceTypeSavedPhotosAlbum**

O usuário faz sua seleção dentre as fotos tiradas mais recentemente.

Figure 11.8 Exemplos de três tipos de origens



O primeiro tipo de origem, `UIImagePickerControllerSourceTypeCamera`, não vai funcionar em um dispositivo que não tenha uma câmera. Portanto, antes de usar esse tipo, você precisa verificar se há uma câmera, enviando a mensagem `isSourceTypeAvailable:` para a classe `UIImagePickerController`:

```
+ (BOOL)isSourceTypeAvailable:(UIImagePickerControllerSourceType)sourceType;
```

O envio dessa mensagem retorna um valor booleano que informa se o dispositivo suporta o tipo de origem passado.

No `BNRDetailViewController.m`, localize o stub de `takePicture:`. Adicione o seguinte código para criar o seletor de imagens e defina sua `sourceType`.

```
- (IBAction)takePicture:(id)sender
{
    UIImagePickerController *imagePickerController =
        [[UIImagePickerController alloc] init];

    // If the device has a camera, take a picture, otherwise,
    // just pick from photo library
    if ([UIImagePickerController
        isSourceTypeAvailable:UIImagePickerControllerSourceTypeCamera]) {
        imagePickerController.sourceType = UIImagePickerControllerSourceTypeCamera;
    } else {
        imagePickerController.sourceType = UIImagePickerControllerSourceTypePhotoLibrary;
    }
}
```

## Configuração do delegate do seletor de imagens

Além do tipo de origem, a instância de `UIImagePickerController` precisa de um delegate. Quando o usuário seleciona uma imagem da interface de `UIImagePickerController`, o delegate recebe a mensagem `imagePickerController:didFinishPickingMediaWithInfo:`. (Se o usuário tocar no botão de cancelar, o delegate receberá a mensagem `imagePickerControllerDidCancel:`.)

O delegate do seletor de imagens será a instância de `BNRDetailViewController`. No `BNRDetailViewController.m`, declare que `BNRDetailViewController` está em conformidade com os protocolos `UINavigationControllerDelegate` e `UIImagePickerControllerDelegate`.

```
@interface BNRDetailViewController ()  
    <UINavigationControllerDelegate, UIImagePickerControllerDelegate>
```

Por que **UINavigationControllerDelegate**? A propriedade delegate de **UIImagePickerController** é, na verdade, herdada de sua superclasse, **UINavigationController**, e embora **UIImagePickerController** possua seu próprio protocolo delegate, a propriedade delegate herdada é declarada para apontar para um objeto que está em conformidade com **UINavigationControllerDelegate**.

No **BNRDetailViewController.m**, adicione o seguinte código a **takePicture:** para definir a instância de **BNRDetailViewController** como o delegate do seletor de imagens.

```
- (IBAction)takePicture:(id)sender  
{  
    UIImagePickerController *imagePicker =  
        [[UIImagePickerController alloc] init];  
  
    // If the device has a camera, take a picture, otherwise,  
    // just pick from photo library  
    if ([UIImagePickerController  
        isSourceTypeAvailable:UIImagePickerControllerSourceTypeCamera]) {  
        imagePicker.sourceType = UIImagePickerControllerSourceTypeCamera;  
    } else {  
        imagePicker.sourceType = UIImagePickerControllerSourceTypePhotoLibrary;  
    }  
  
    imagePicker.delegate = self;  
}
```

## Apresentação do seletor de imagens modalmente

Assim que a **UIImagePickerController** tiver um tipo de origem e um delegate, é hora de obter sua view na tela. Diferentemente de outras subclasses de **UIViewController** que você usou, uma instância de **UIImagePickerController** é apresentada *modalmente*. Um *controlador de visão modal* ocupa a tela inteira até que seu trabalho seja finalizado.

Para apresentar um controlador de visão de forma modal, envie **presentViewController:animated:completion:** para **UIViewController**, cuja view encontra-se na tela. O controlador de visão a ser apresentado é passado para a tela e a visão do controlador de visão desliza da parte inferior para cima. (Você terá mais informações sobre como apresentar controladores de visão modais no Chapter 17.)

No **BNRDetailViewController.m**, adicione código ao final do método **takePicture:** para apresentar a **UIImagePickerController**.

```
imagePicker.delegate = self;  
  
// Place image picker on the screen  
[self presentViewController:imagePicker animated:YES completion:nil];  
}
```

(O terceiro argumento, **completion:**, espera um bloco. Você saberá mais sobre blocos de conclusão no Chapter 17.)

Agora, você pode compilar e executar o aplicativo. Selecione uma **BNRItem** para ver os detalhes e, em seguida, toque no botão de câmera na **UIToolbar**. A interface da **UIImagePickerController** aparecerá na tela (Figure 11.9), e você pode tirar uma foto ou escolher uma imagem existente, caso seu dispositivo não tenha câmera.

(Se você estiver trabalhando no simulador, pode abrir o **Safari** no simulador e navegar até uma página com uma imagem. Clique na imagem e mantenha-a pressionada, depois selecione **Save Image** para salvá-la na biblioteca de fotos do simulador. Em seguida, essa imagem será exibida no seletor de imagens. Contudo, o simulador pode ser complicado; talvez você tenha que tentar com imagens diferentes, antes de alguma delas ser realmente salva na biblioteca.)

Figure 11.9 Interface de pré-visualização **UIImagePickerController**

## Salvamento da imagem

A seleção de uma imagem dispensa a **UIImagePickerController** e faz você retornar à visão de detalhes. No entanto, você não possui uma referência para a foto, já que o seletor de imagens foi dispensado. Para corrigir isso, você terá de implementar o método delegate **imagePickerController:didFinishPickingMediaWithInfo:**. Essa mensagem é enviada para o delegate do seletor de imagens quando uma foto é selecionada.

No **BNRDetailViewController.m**, implemente esse método para colocar a imagem na **UIImageView** e então enviar uma mensagem para dispensar o seletor de imagens.

```
- (void)imagePickerController:(UIImagePickerController *)picker
didFinishPickingMediaWithInfo:(NSDictionary *)info
{
    // Get picked image from info dictionary
    UIImage *image = info[UIImagePickerControllerOriginalImage];

    // Put that image onto the screen in our image view
    self.imageView.image = image;

    // Take image picker off the screen -
    // you must call this dismiss method
    [self dismissViewControllerAnimated:YES completion:nil];
}
```

Compile e execute o aplicativo novamente. Tire (ou selecione) uma foto. O seletor de imagens é dispensado, e você retorna para a view de **BNRDetailViewController**.

Você poderia ter centenas de itens e cada um deles poderia ter uma imagem grande associada a eles. Não há problema em manter centenas de instâncias de **BNRItem** na memória. Manter centenas de imagens na memória seria uma má ideia neste caso: primeiro, você recebe um aviso de pouca memória. Então, se o espaço ocupado de memória do aplicativo aumentar, o sistema operacional o encerrará. A solução, que você implementará na próxima seção, é armazenar as imagens no disco e buscá-las na RAM apenas quando necessário. Essa busca será feita por uma nova classe: **BNRImageStore**. Quando a **BNRImageStore** receber uma notificação de pouca memória, ela esvaziará o cache para liberar a memória que as imagens buscadas estavam ocupando.

## Criação de BNRIImageStore

O armazenamento de imagens conterá as fotos que o usuário tirar. No Chapter 18, você vai fazer com que as instâncias de **BNRItem** escrevam suas variáveis de instância em um arquivo, que serão lidas quando o aplicativo for iniciado. No entanto, já que as imagens tendem a ser muito grandes, é uma boa ideia mantê-las separadas dos outros dados. O armazenamento de imagens vai obter e armazenar as imagens em cache, conforme forem necessárias. Ele também será capaz de esvaziar o cache se o dispositivo ficar com pouca memória.

Crie uma nova subclasse **NSObject** chamada **BNRIImageStore**. Abra o **BNRIImageStore.h** e crie sua interface:

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

@interface BNRIImageStore : NSObject
+ (instancetype)sharedStore;
- (void)setImage:(UIImage *)image forKey:(NSString *)key;
- (UIImage *)imageForKey:(NSString *)key;
- (void)deleteImageForKey:(NSString *)key;
@end
```

No **BNRIImageStore.m**, adicione uma extensão de classe que declare que uma propriedade manterá as imagens.

```
@interface BNRIImageStore ()
@property (nonatomic, strong) NSMutableDictionary *dictionary;
@end

@implementation BNRIImageStore
```

Como a **BNRItemStore**, a **BNRIImageStore** precisa ser um singleton. No **BNRIImageStore.m**, escreva o código a seguir, para garantir o status de singleton para **BNRIImageStore**.

```
@implementation BNRIImageStore
+ (instancetype)sharedStore
{
    static BNRIImageStore *sharedStore = nil;
    if (!sharedStore) {
        sharedStore = [[self alloc] initPrivate];
    }
    return sharedStore;
}

// No one should call init
- (instancetype)init
{
    @throw [NSEException exceptionWithName:@"Singleton"
                                         reason:@"Use +[BNRIImageStore sharedStore]"
                                         userInfo:nil];
    return nil;
}

// Secret designated initializer
- (instancetype)initPrivate
{
    self = [super init];
    if (self) {
        _dictionary = [[NSMutableDictionary alloc] init];
    }
    return self;
}
```

Depois, implemente os outros três métodos declarados no arquivo de cabeçalho.

```
- (void)setImage:(UIImage *)image forKey:(NSString *)key
{
    [self.dictionary setObject:image forKey:key];
}

- (UIImage *)imageForKey:(NSString *)key
{
    return [self.dictionary objectForKey:key];
}

- (void)deleteImageForKey:(NSString *)key
{
    if (!key) {
        return;
    }
    [self.dictionary removeObjectForKey:key];
}
```

## NSDictionary

Observe que `dictionary` é uma instância de `NSMutableDictionary`. Como um array, um *dicionário* é um objeto de coleção que possui uma versão imutável (`NSDictionary`) e uma versão mutável (`NSMutableDictionary`).

Dicionários e arrays são diferentes na forma como armazenam seus objetos. Um array é uma lista ordenada de ponteiros para objetos, que é acessada por índice. Quando você tem um array, pode pedir pelo objeto no *enésimo* índice:

```
// Put some object at the beginning of an array
[someArray insertObject:someObject atIndex:0];

// Get that same object out
someObject = [someArray objectAtIndex:0];
```

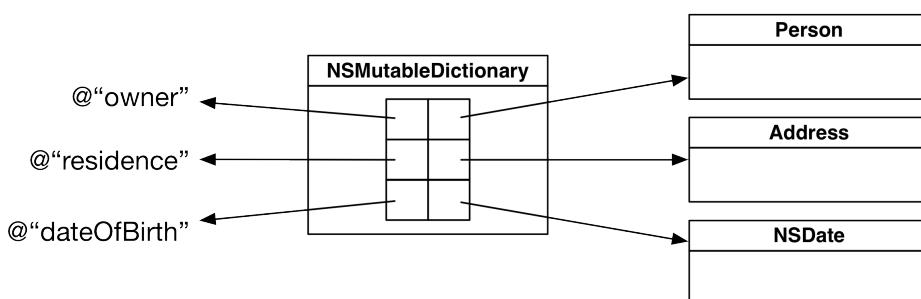
Objetos de um dicionário não são ordenados dentro do conjunto. Portanto, em vez de acessar as entradas com índice, você usa uma *chave*. A chave geralmente é uma instância de `NSString`.

```
// Add some object to a dictionary for the key "MyKey"
[someDictionary setObject:someObject forKey:@"MyKey"];

// Get that same object out
someObject = [someDictionary objectForKey:@"MyKey"];
```

Nós chamamos cada entrada de um dicionário de *par chave-valo*r. O *valor* é o objeto sendo armazenado na coleção, e a *chave* é um valor único (geralmente uma string), que você usa para armazenar e recuperar o valor posteriormente. (Em ambientes de desenvolvimento, um dicionário é chamado de *tabela de dispersão*, *tabela hash* ou *array associativo*, mas ainda usamos o termo par chave-valo para falar sobre as informações que armazenam.)

Figure 11.10 Diagrama de `NSDictionary`



Existem diversos tipos de uso da **NSDictionary**. Os dois mais comuns são estruturas de dados flexíveis e tabelas de pesquisa.

Primeiro, vamos falar das estruturas de dados flexíveis. Geralmente, quando você quer representar um objeto de modelo, você cria uma subclasse de **NSObject** e fornece a ela variáveis de instância apropriadas. Por exemplo, uma classe **Person** teria variáveis de instância como `firstName`, `age` e outras coisas que você espera que uma pessoa real tenha. Uma instância de **NSDictionary** também pode ser usada para representar um objeto de modelo. No exemplo da pessoa, ela poderia conter valores para as chaves `firstName` (nome), `age` (idade) e outras coisas que você espera que uma pessoa real tenha.

A diferença é que a classe **Person** requer que você defina exatamente o que é a **Person**, não sendo possível adicionar, remover ou alterar a composição estrutural de uma pessoa. Com uma **NSDictionary**, se você deseja adicionar `address` (endereço) a “Person” (Pessoa), basta adicionar um valor para a chave `address`.

Este não é um endosso para usar **NSDictionary** para representar todos os objetos – a maioria dos objetos precisa ter uma definição rígida, regras para a maneira como armazenam, salvam e carregam dados, e comportamento que não seja apenas armazenar dados. Geralmente, isso significa definir uma classe personalizada, como **BNRItem**. Contudo, a **NSDictionary** costuma ser usada para representar dados passados para ou retornados de um método, que podem ter uma estrutura diferente dependendo das opções especificadas. Por exemplo, o método `delegate` da **UIImagePickerController** entrega a você a **NSDictionary**, que pode conter uma imagem ou um vídeo, de acordo com a maneira como você configurou o seletor de imagens. O dicionário também pode conter metadados relacionados a essa imagem ou vídeo.

A outra utilização comum de **NSDictionary** é a criação de tabelas de pesquisa. Alguma vez em sua carreira de programador, você provavelmente fez algo assim:

```
- (void)changeCharacterClass:(id)sender
{
    NSString *enteredText = textField.text;
    CharacterClass *cc = nil;

    if ([enteredText isEqualToString:@"Warrior"]) {
        cc = knight;
    } else if ([enteredText isEqualToString:@"Mage"]) {
        cc = wizard;
    } else if ([enteredText isEqualToString:@"Thief"]) {
        cc = rogue;
    }

    character.characterClass = cc;
}
```

Um dicionário pode resolver o problema de criação de enormes instruções `if-elses` ou `switch` predeterminando o mapeamento entre dois objetos. Continuando com o exemplo mais “nerd” que existe, uma **NSDictionary** poderia ser inicializada desta maneira:

```
NSMutableDictionary *lookup = [[NSMutableDictionary alloc] init];
[lookup setObject:knight forKey:@"Warrior"];
[lookup setObject:wizard forKey:@"Mage"];
[lookup setObject:rogue forKey:@"Thief"];
```

sendo possível, então, alterar o método `changeCharacterClass:` para algo muito mais claro:

```
- (void)changeCharacterClass:(id)sender
{
    character.characterClass = [lookup objectForKey:textField.text];
}
```

A vantagem é que, com essa abordagem, você não precisa fixar no código todas as possibilidades, mas poderia armazená-las em um arquivo de dados, obtê-las de um servidor em algum lugar ou adicioná-las dinamicamente com alguma ação do usuário. É assim que a **BNRImageStore** funciona: uma chave será gerada para mapear uma imagem e usada para pesquisar essa imagem posteriormente.

Ao usar um dicionário, só pode haver um objeto para cada chave. Se você adicionar um objeto ao dicionário com uma chave igual à chave de um objeto já presente nele, o objeto anterior é removido. Se precisar armazenar

vários objetos para uma chave, é possível colocá-los em um array e adicionar este array como valor ao dicionário.

Os dicionários, assim como os arrays, podem ser criados usando-se uma sintaxe abreviada. A sintaxe abreviada para a criação de dicionários usa chaves (@{}), diferentemente dos colchetes que a **NSArray** usa (@[]). Ao inicializar um dicionário usando uma sintaxe abreviada, cada par chave-valor é separado por uma vírgula (,),. O dois pontos (:) é colocado entre a chave e seu valor.

```
NSDictionary *dictionary = @{@"key": object, @"anotherKey": anotherObject};
```

Os dicionários também têm uma sintaxe abreviada para recuperar objetos:

```
id object = dictionary[@"key"];
// same as
id object = [dictionary objectForKey:@"key"];
```

Se você possui uma **NSMutableDictionary**, pode definir o objeto para uma chave com sintaxe abreviada:

```
dictionary[@"key"] = object;
// same as
[dictionary setObject:object forKey:@"key"];
```

Atualize o armazenamento de imagens para usar a forma abreviada de acesso e modificação de dicionários.

```
- (void)setImage:(UIImage *)image forKey:(NSString *)key
{
    [self.dictionary setObject:image forKey:key];
    self.dictionary[key] = image;
}

- (UIImage *)imageForKey:(NSString *)key
{
    return [self.dictionary objectForKey:key];
    return self.dictionary[key];
}
```

Finalmente, observe que o gerenciamento de memória de um dicionário é como aquele de um array. Sempre que você adiciona um objeto ao dicionário, ele tem a propriedade desse objeto, e sempre que você remove um objeto do dicionário, ele libera a propriedade.

## Criação e uso de chaves

Quando uma imagem for adicionada ao armazenamento, ela será colocada em um dicionário, em uma chave única, e o objeto de **BNRItem** relacionado receberá essa chave. Quando a **BNRDetailViewController** quiser uma imagem do armazenamento, pedirá a chave à sua variável **item** e procurará pela imagem no dicionário. Adicione uma propriedade ao **BNRItem.h** para armazenar a chave.

```
@property (nonatomic, readonly, strong) NSDate *dateCreated;
```

```
@property (nonatomic, copy) NSString *itemKey;
```

As chaves de imagens devem ser únicas para que seu dicionário funcione. Embora existam várias maneiras de elaborar uma string única, você usará mecanismos de Cocoa Touch para criar identificadores universais exclusivos (UUIDs), também conhecidos como identificadores globais exclusivos (GUIDs). Objetos do tipo **NSUUID** representam um UUID e são gerados usando o tempo, um contador e um identificador de hardware, que geralmente é o endereço MAC da placa Wi-Fi. Quando representados na forma de string, os UUIDs ficarão mais ou menos assim:

4A73B5D2-A6F4-4B40-9F82-EA1E34C1DC04

Importe **BNRImageStore.h**, na parte superior de **BNRDetailViewController.m**.

```
#import "BNRDetailViewController.h"
#import "BNRItem.h"
#import "BNRImageStore.h"
```

No **BNRItem.m**, modifique **init** para gerar um UUID e defini-lo como a **itemKey**.

```

- (instancetype)initWithItemName:(NSString *)name
                           valueInDollars:(int)value
                            serialNumber:(NSString *)sNumber
{
    // Call the superclass's designated initializer
    self = [super init];
    // Did the superclass's designated initializer succeed?
    if (self) {
        // Give the instance variables initial values
        _itemName = name;
        _serialNumber = sNumber;
        _valueInDollars = value;
        // set _dateCreated to the current date and time
        _dateCreated = [[NSDate alloc] init];

        // Create an NSUUID object - and get its string representation
        NSUUID *uuid = [[NSUUID alloc] init];
        NSString *key = [uuid UUIDString];
        _itemKey = key;
    }
    // Return the address of the newly initialized object
    return self;
}

```

Em seguida, no `BNRDetailViewController.m`, atualize `imagePickerController:didFinishPickingMediaWithInfo:` para armazenar a imagem na `BNRImageStore`.

```

- (void)imagePickerController:(UIImagePickerController *)picker
didFinishPickingMediaWithInfo:(NSDictionary *)info
{
    UIImage *image = info[UIImagePickerControllerOriginalImage];

    // Store the image in the BNRImageStore for this key
    [[BNRImageStore sharedStore] setImage:image
                                    forKey:self.item.itemKey];

    imageView.image = image;
    [self dismissViewControllerAnimated:YES completion:nil];
}

```

Toda vez que uma imagem for capturada, ela será adicionada ao armazenamento. Tanto a `BNRImageStore` quanto a `BNRItem` reconhecerão a chave da imagem, portanto, ambas poderão ter acesso a ela, quando necessário.

Da mesma forma, quando um item é excluído, você precisa excluir sua imagem do armazenamento de imagens. Na parte superior do `BNRItemStore.m`, importe o cabeçalho da `BNRImageStore` e adicione o seguinte código ao `removeItem::`:

```

#import "BNRImageStore.h"

@implementation BNRItemStore

- (void)removeItem:(BNRItem *)item
{
    NSString *key = item.itemKey;
    if (key) {
        [[BNRImageStore sharedStore] deleteImageForKey:key];
    }

    [self.privateItems removeObjectIdenticalTo:item];
}

```

Você pode estar se perguntando: “por que não fornecer a `BNRItem` um ponteiro para a imagem? Afinal, não é o ponteiro para a imagem uma forma mais direta de fazer a referência à imagem?” Mesmo sendo isso correto, você deverá considerar o que acontece quando começar a salvar os itens e as imagens correspondentes no sistema de arquivos no Chapter 18.

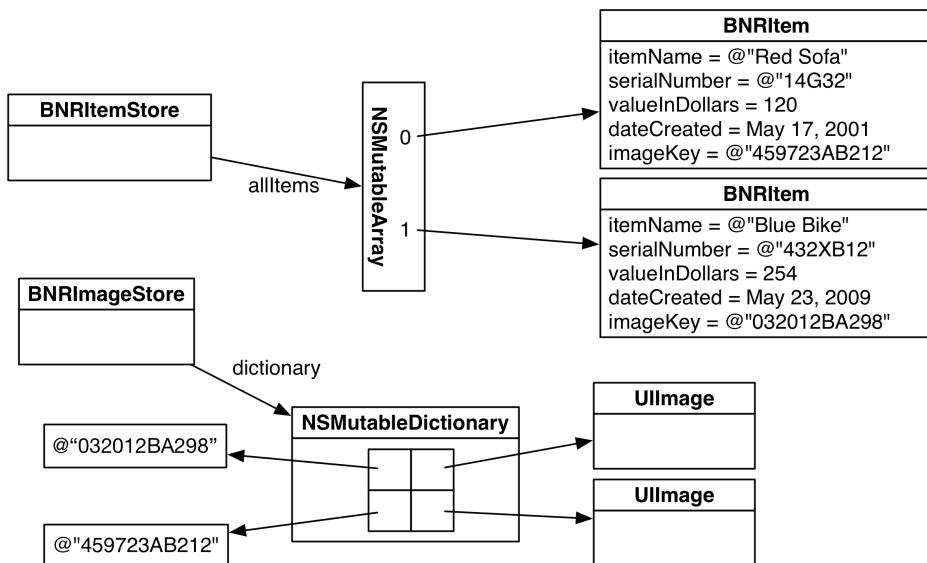
Quando uma `UIImage` é criada pela primeira vez, ela fica na memória, em um endereço específico. Um ponteiro mantém esse endereço, assim você pode consultar a imagem novamente. Contudo, na próxima vez que o

aplicativo for executado, a imagem não estará no mesmo endereço de memória, portanto, você não poderá usar o mesmo ponteiro para acessá-la. Em vez disso, a chave será usada para nomear o arquivo de imagem no sistema de arquivos e cada **BNRItem** manterá sua chave. Quando você quiser carregar a imagem novamente para a memória, a **BNRImageStore** usará a `itemKey` de uma **BNRItem** para encontrar o arquivo de imagem no sistema de arquivos, carregá-lo na memória e fazer um ponteiro retornar para a nova instância de **UIImage**. Portanto, a chave é uma maneira persistente de consultar uma imagem.

## Finalização de BNRImageStore

Agora que **BNRImageStore** pode armazenar imagens, e as instâncias de **BNRItem** têm uma chave para obter esta imagem (Figure 11.11), você precisa ensinar à **BNRDetailViewController** como capturar a imagem para a **BNRItem** selecionada e colocá-la em sua `imageView`.

Figure 11.11 Cache



A **BNRDetailViewController** de view aparecerá em dois momentos: quando o usuário tocar em uma linha em **BNRItemsViewController** e quando **UIImagePickerController** for dispensada. Em ambas as situações, a `imageView` deve ser preenchida com a imagem da **BNRItem** sendo exibida.

No `BNRDetailViewController.m`, adicione um código a `viewWillAppear:` para fazer isso.

```
- (void)viewWillAppear:(BOOL)animated
{
    [super viewWillAppear:animated];
    self.nameField.text = item.itemName;
    self.serialNumberField.text = item.serialNumber;
    self.valueField.text = [NSString stringWithFormat:@"%d",
                           item.valueInDollars];

    static NSDateFormatter *dateFormatter = nil;
    if (!dateFormatter) {
        dateFormatter = [[NSDateFormatter alloc] init];
        dateFormatter.dateStyle = NSDateFormatterMediumStyle;
        dateFormatter.timeStyle = NSDateFormatterNoStyle;
    }

    self.dateLabel.text = [dateFormatter stringFromDate:item.dateCreated];

    NSString *itemKey = self.item.itemKey;

    if (itemKey) {
        // Get image for image key from image store
        UIImage *imageToDisplay =
            [[BNRImageStore sharedStore] imageForKey:itemKey];

        // Use that image to put on the screen in imageView
        self.imageView.image = imageToDisplay;
    } else {
        // Clear the imageView
        self.imageView.image = nil;
    }
}
```

Observe que se não houver nenhuma imagem no armazenamento de imagens para essa chave (ou não houver chave para esse item), o ponteiro para a imagem será `nil`. Quando a imagem é `nil`, a `UIImageView` não exibe uma imagem.

Compile e execute o aplicativo. Crie uma `BNRItem` e selecione-a na `UITableView`. Em seguida, toque no botão de câmera e tire uma foto. A imagem aparecerá como deve.

Há outro detalhe para se prestar atenção: se você selecionar uma nova imagem para `BNRItem`, a antiga ainda estará na `BNRImageStore`. No início de `imagePickerController:didFinishPickingMediaWithInfo:` no `BNRDetailViewController.m`, adicione algum código para instruir `BNRImageStore` a remover a imagem antiga.

```
- (void)imagePickerController:(UIImagePickerController *)picker
    didFinishPickingMediaWithInfo:(NSDictionary *)info
{
    NSString *oldKey = self.item.itemKey;

    // Did the item already have an image?
    if (oldKey) {
        // Delete the old image
        [[BNRImageStore sharedStore] deleteImageForKey:oldKey];
    }

    UIImage *image = info[UIImagePickerControllerOriginalImage];
```

Compile e execute o aplicativo novamente. O comportamento deve permanecer o mesmo, mas os benefícios para a memória são significativos.

## Dispensando o teclado

Quando o teclado aparece na tela na visão de detalhes de itens, ele oculta a `imageView` de `BNRDetailViewController`. Isso é irritante quando se está tentando ver uma imagem; portanto, você implementará o método delegado `textFieldShouldReturn:` para que o campo de texto desista de seu status de primeiro respondente, de modo a dispensar o teclado quando a tecla de retorno for tocada. (Foi por isso que você conectou os outlets de delegate anteriormente.) Mas primeiro, em `BNRDetailViewController.m`, faça com que `BNRDetailViewController` esteja em conformidade com o protocolo `UITextFieldDelegate`.

```
@interface BNRDetailViewController : UIViewController
<UINavigationControllerDelegate, UIImagePickerControllerDelegate,
UITextFieldDelegate>
```

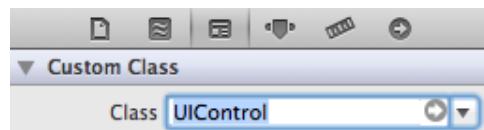
No `BNRDetailViewController.m`, implemente `textFieldShouldReturn:`.

```
- (BOOL)textFieldShouldReturn:(UITextField *)textField
{
    [textField resignFirstResponder];
    return YES;
}
```

Seria interessante dispensar o teclado também se o usuário tocar em algum outro lugar da view de `BNRDetailViewController`. Você pode dispensar o teclado enviando à view a mensagem `endEditing:`, que fará com que o campo de texto (como uma subvisão da view) desista de ser o primeiro respondente. Agora, vamos descobrir como fazer com que a visão, quando tocada, envie a mensagem.

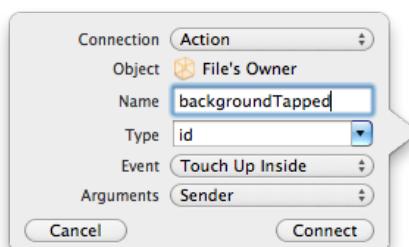
Você já viu que classes como `UIButton`, quando tocadas, podem enviar uma mensagem de ação a um destino. Botões herdam esse comportamento destino-ação de sua superclasse `UIControl`. Você vai alterar a view de `BNRDetailViewController` de uma instância de `UIView` para uma instância de `UIControl`, para que possa lidar com eventos de toque.

No `BNRDetailViewController.xib`, selecione o objeto principal da View. Abra o inspetor de identidade e mude a classe da view para `UIControl` (Figure 11.12).

Figure 11.12 Alteração da classe da visão de **BNRDetailViewController**

Em seguida, abra o `BNRDetailViewController.m` no editor assistente. Pressionando a tecla Control, arraste a view (agora uma `UIControl`) para a implementação de `BNRDetailViewController`. Quando a janela pop-up for exibida, selecione Action no menu pop-up Connection. Observe que a interface dessa janela pop-up é um pouco diferente daquela que você viu ao criar e conectar a `UIBarButtonItem`. Uma `UIBarButtonItem` é uma versão simplificada da `UIControl` – só envia uma mensagem de ação ao seu destino quando é tocada. Uma `UIControl`, por outro lado, pode enviar mensagens de ação em resposta a uma variedade de eventos.

Portanto, você deve escolher o tipo de evento adequado para disparar a mensagem de ação que está sendo enviada. Nesse caso, você quer que a mensagem de ação seja enviada quando o usuário toca na visão. Configure esta janela pop-up de forma que seja exibida como na Figure 11.13 e clique em Connect.

Figure 11.13 Configuração de uma ação de **UIControl**

Isso criará um método stub no `BNRDetailViewController.m`. Insira o código a seguir neste método.

```
- (IBAction)backgroundTapped:(id)sender
{
    [self.view endEditing:YES];
}
```

Compile e execute o aplicativo e teste as duas maneiras de dispensar o teclado.

## Desafio de bronze: edição de imagens

`UIImagePickerController` tem uma interface integrada para editar uma imagem depois de ter sido selecionada. Permita que o usuário edite a imagem e use a imagem editada, em vez da imagem original, em `BNRDetailViewController`.

## Desafio de prata: remoção de imagens

Adicione um botão que limpe a imagem para um item.

## Desafio de ouro: sobreposição de câmera

Uma `UIImagePickerController` tem uma propriedade `cameraOverlayView`. Faça com que a apresentação da `UIImagePickerController` mostre um retículo no meio da área de captura de imagem.

## Para os mais curiosos: navegação por arquivos de implementação

Seus dois controladores de visão têm muitos métodos em seus arquivos de implementação. Para ser um desenvolvedor de iOS eficiente, você deve ser capaz de acessar o código que está buscando de forma rápida e

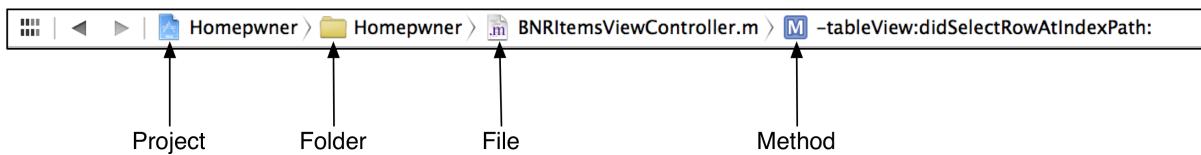
fácil. A barra de salto do editor de origem no Xcode é uma ferramenta que está à sua disposição para ajudá-lo (Figure 11.14).

Figure 11.14 Barra de salto do editor de origem



A barra de salto mostra exatamente onde você está no projeto (e também onde o cursor está em determinado arquivo). A Figure 11.15 mostra a barra de salto em detalhes.

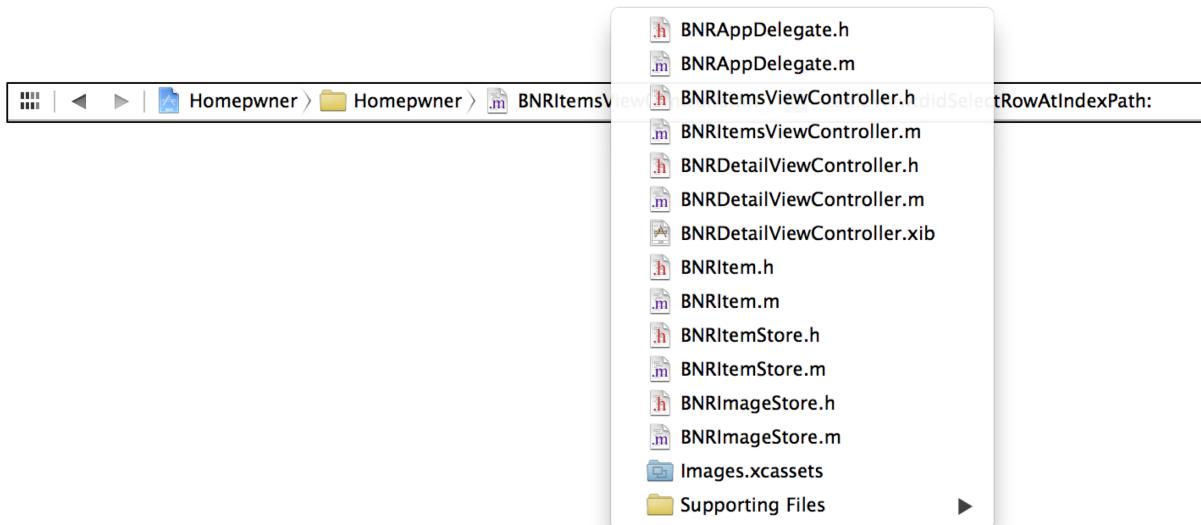
Figure 11.15 Detalhes da barra de salto



A navegação de trilha estrutural da barra de salto segue a hierarquia da navegação por projetos. Se você clicar em uma das seções, verá um popover dessa seção na hierarquia de projetos, e deste lugar você pode navegar por outras partes do projeto.

A Figure 11.16 exibe o popover de arquivos no aplicativo Homepwner.

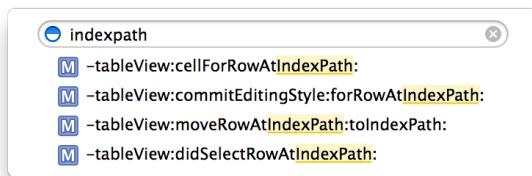
Figure 11.16 Popover de arquivos



Talvez seja mais útil a capacidade de navegar facilmente por um arquivo de implementação. Se você clicar no último elemento da trilha estrutural, verá um popover com o conteúdo do arquivo, incluindo todos os métodos implementados nesse arquivo.

Enquanto o popover ainda está visível, você pode começar a digitar para filtrar os itens na lista. A qualquer momento, é possível usar as teclas para cima e para baixo e pressionar a tecla Enter para saltar para esse método no código. A Figure 11.17 mostra o que você obtém quando pesquisa por “indexPath” no BNRItemsViewController.m.

Figure 11.17 Popover de arquivos com pesquisa “indexpath”



## #pragma mark

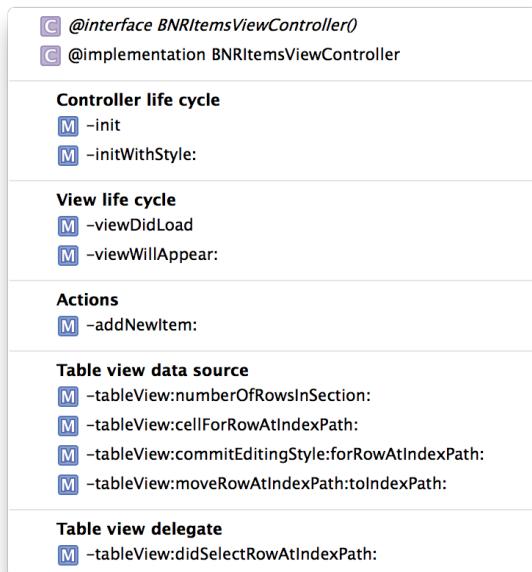
À medida que suas classes ficam maiores, pode ser mais difícil encontrar o método que você está buscando, quando ele está perdido em uma longa lista de métodos. Uma boa maneira de organizar seus métodos e arrumar a bagunça é usar a diretiva de pré-processador `#pragma mark`.

```
#pragma mark - View life cycle
- (void)viewDidLoad {...}
- (void)viewWillAppear:(BOOL)animated {...}

#pragma mark - Actions
- (void)addNewItem:(id)sender {...}
```

Adicionar `#pragma marks` ao seu código não altera nada no código, em vez disso, ajuda o Xcode a entender como você gostaria de organizar visualmente seus métodos. Você pode ver os resultados da adição deles abrindo o item do arquivo atual na barra de salto. A Figure 11.18 mostra os resultados de um `BNRItemsViewController.m` bem organizado.

Figure 11.18 Popover de arquivos com #pragma marks



Duas `#pragma marks` úteis são o divisor e o rótulo.

```
// This is a divider
#pragma mark -

// This is a label
#pragma mark My Awesome Methods

// They can be combined as well
#pragma mark - My Awesome Methods
```

Ao utilizar a diretiva `#pragma mark`, você força a si próprio a organizar o código. Se tudo sair bem, seu código ficará mais legível e mais fácil de trabalhar quando você, inevitavelmente, precisar revisitar o código. Depois de fazer isso repetidamente, você vai adquirir hábitos que lhe ajudarão a navegar pelo banco de códigos.

## Para os mais curiosos: gravação de vídeo

Depois que você entendeu como usar **UIImagePickerController** para tirar fotos, passar para a gravação de vídeos é simples. Lembre-se que um controlador de seleção de imagem tem uma propriedade **sourceType** que determina se a imagem é proveniente da câmera, da biblioteca de fotos ou de um álbum de fotos salvas. Controladores de seleção de imagem também têm uma propriedade **mediaTypes**, que é um array de strings que contém identificadores dos tipos de mídia que podem ser selecionados a partir de três tipos de origens.

Há dois tipos de mídia que uma **UIImagePickerController** pode selecionar: imagens estáticas e vídeo. Por padrão, o array de **mediaTypes** contém apenas a string da constante **KUTTypeImage**. Assim, se você não mudar a propriedade **mediaTypes** de um controlador de seleção de imagem, a câmera permitirá que o usuário tire apenas fotos estáticas, e a biblioteca de fotos e o álbum de fotos salvas exibirão apenas imagens.

Adicionar a capacidade de gravar vídeos ou escolher um vídeo do disco é tão simples quanto adicionar a string da constante **KUTTypeMovie** ao array de **mediaTypes**. No entanto, nem todos os dispositivos suportam vídeo através de **UIImagePickerController**. Assim como o método de classe **isSourceTypeAvailable**: permite que você determine se o dispositivo tem câmera, o método **availableMediaTypesForSourceType**: verifica se a câmera pode capturar vídeo. Para configurar um controlador de seleção de imagem que pode gravar vídeo ou tirar fotos estáticas, você escreveria o seguinte código:

```
UIImagePickerController *ipc = [[UIImagePickerController alloc] init];
NSArray *availableTypes = [UIImagePickerController
    availableMediaTypesForSourceType:UIImagePickerControllerSourceTypeCamera];
ipc.mediaTypes = availableTypes;
ipc.sourceType = UIImagePickerControllerSourceTypeCamera;
ipc.delegate = self;
```

Quando esta interface do controlador de seleção de imagem for apresentada ao usuário, haverá um botão que permite escolher entre a câmera de imagens estáticas e o gravador de vídeo. Se o usuário escolher gravar um vídeo, você precisa lidar com isso no método de delegate **UIImagePickerController** da **imagePickerController:didFinishPickingMediaWithInfo**:

Ao lidar com imagens estáticas, o dicionário **info** que é passado como um argumento contém a imagem completa como um objeto de **UIImage**. No entanto, não há nenhuma classe de “**UIVideo**”. (Carregar um vídeo inteiro na memória de uma vez seria difícil com as limitações da memória de dispositivos iOS.) Portanto, os vídeos são gravados em disco em um diretório temporário. Quando o usuário finaliza a gravação do vídeo, **imagePickerController:didFinishPickingMediaWithInfo**: é enviado ao delegate do controlador de seleção de imagem e o caminho do vídeo no disco encontra-se no dicionário **info**. Você pode obter o caminho assim:

```
- (void)imagePickerController:(UIImagePickerController *)picker
didFinishPickingMediaWithInfo:(NSDictionary *)info
{
    NSURL *mediaURL = info[UIImagePickerControllerMediaURL];
}
```

Você aprenderá sobre o sistema de arquivos no Chapter 18, mas o que você deve saber agora é que o diretório temporário não é um lugar seguro para armazenar o vídeo. Ele precisa ser transferido para outro local.

```
- (void)imagePickerController:(UIImagePickerController *)picker
didFinishPickingMediaWithInfo:(NSDictionary *)info
{
    NSURL *mediaURL = info[UIImagePickerControllerMediaURL];
    if (mediaURL) {

        // Make sure this device supports videos in its photo album
        if (UIVideoAtPathIsCompatibleWithSavedPhotosAlbum([mediaURL path])) {

            // Save the video to the photos album
            UISaveVideoAtPathToSavedPhotosAlbum([mediaURL path], nil, nil, nil);

            // Remove the video from the temporary directory
            [[NSFileManager defaultManager] removeItemAtPath:[mediaURL path]
                                            error:nil];
        }
    }
}
```

Isso é realmente tudo. Há apenas uma situação que requer informações adicionais: suponha que você queira restringir o usuário a escolher *somente* vídeos. Restringir o usuário a imagens é simples (deixe `mediaTypes` como padrão). Permitir que o usuário escolha entre imagens e vídeos é igualmente simples (passe o valor de retorno de `availableMediaTypesForSourceType:`). No entanto, para permitir somente vídeo, você precisa fazer algumas coisas. Primeiro, você deve certificar-se de que o dispositivo suporta vídeo e, em seguida, deve definir a propriedade `mediaTypes` para um array que contenha apenas o identificador para vídeo.

```
NSArray *availableTypes = [UIImagePickerController
    availableMediaTypesForSourceType:UIImagePickerControllerSourceTypeCamera];

if ([availableTypes containsObject:(__bridge NSString *)kUTTypeMovie]) {
    [ipc setMediaTypes:@[(__bridge NSString *)kUTTypeMovie]];
}
```

Quer saber por que `kUTTypeMovie` é convertida para uma `NSString`? Essa constante é declarada como:

```
const CFStringRef kUTTypeMovie;
```

Se você compilar esse código, ele falhará, e o compilador reclamará que nunca ouviu falar de `kUTTypeMovie`. Por mais estranho que pareça, tanto `kUTTypeMovie` quanto `kUTTypeImage` são declaradas e definidas em outro framework: **MobileCoreServices**. Você precisa adicionar este framework explicitamente e importar seu arquivo de cabeçalho para o projeto, de modo a usar estas duas constantes.



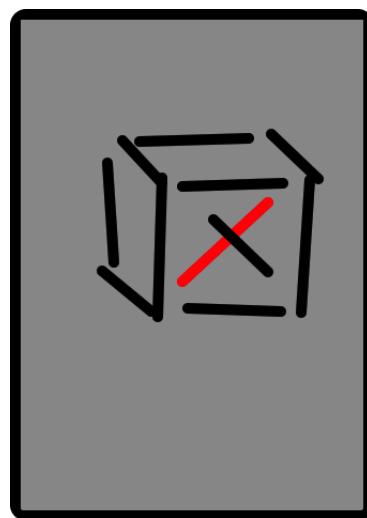
# 12

## Eventos de toque e UIResponder

Nos próximos três capítulos, você vai deixar de trabalhar com o Homepwner e vai criar um novo aplicativo chamado TouchTracker para saber mais sobre eventos de toque e gestos, além de depurar aplicativos.

Neste capítulo, você criará uma visão que permite que o usuário trace linhas, arrastando o dedo pela visão (Figure 12.1). Usando o multitoque, o usuário será capaz de traçar mais de uma linha por vez.

Figure 12.1 Um programa de desenho



### Eventos de toque

Como uma subclasse de **UIResponder**, a **UIView** pode sobrescrever quatro métodos para lidar com os quatro eventos de toque distintos:

- um dedo (ou dedos) toca a tela
  - `(void)touchesBegan:(NSSet *)touches  
withEvent:(UIEvent *)event;`
- um dedo (ou dedos) se move pela tela (esta mensagem é enviada repetidamente enquanto o dedo se move)
  - `(void)touchesMoved:(NSSet *)touches  
withEvent:(UIEvent *)event;`
- um dedo (ou dedos) é retirado da tela
  - `(void)touchesEnded:(NSSet *)touches  
withEvent:(UIEvent *)event;`
- um evento do sistema, como uma chamada telefônica, interrompe um toque antes de este terminar
  - `(void)touchesCancelled:(NSSet *)touches  
withEvent:(UIEvent *)event;`

Quando um dedo toca a tela, uma instância de **UITouch** é criada. A **UIView** que este dedo tocou recebe uma mensagem **touchesBegan:withEvent:** e **UITouch** encontra-se na **NSSet** de **touches**.

À medida que o dedo se move pela tela, o objeto de toque é atualizado para conter a localização atual do dedo na tela. Em seguida, a mesma **UIView** em que o toque começou recebe a mensagem **touchesMoved:withEvent:**. A **NSSet** que é passada como um argumento para este método contém a mesma **UITouch** que foi criada, originalmente, quando o dedo que ela representa tocou a tela.

Quando um dedo é removido da tela, o objeto de toque é atualizado uma última vez, de modo a conter a localização atual do dedo e a visão em que o dedo começou receber a mensagem **touchesEnded:withEvent:**. Depois que esse método termina a execução, o objeto **UITouch** é destruído.

A partir desta informação, podemos chegar a algumas conclusões sobre como os objetos de toque funcionam:

- Uma **UITouch** corresponde a um dedo na tela. Este objeto de toque existe enquanto o dedo estiver na tela e sempre contém a posição atual do dedo na tela.
- A visão em que o dedo começou receberá todas as mensagens de evento de toque para aquele dedo, não importa o que aconteça. Se o dedo se mover para fora do **UIView** da frame em que começou, essa visão ainda receberá as mensagens **touchesMoved:withEvent:** e **touchesEnded:withEvent:**. Dessa forma, se um toque começa em uma visão, então essa visão possui o toque enquanto ele durar.
- Você não precisa – nem nunca deve – manter uma referência para um objeto **UITouch**. O aplicativo lhe dará acesso a um objeto de toque quando ele muda de estado.

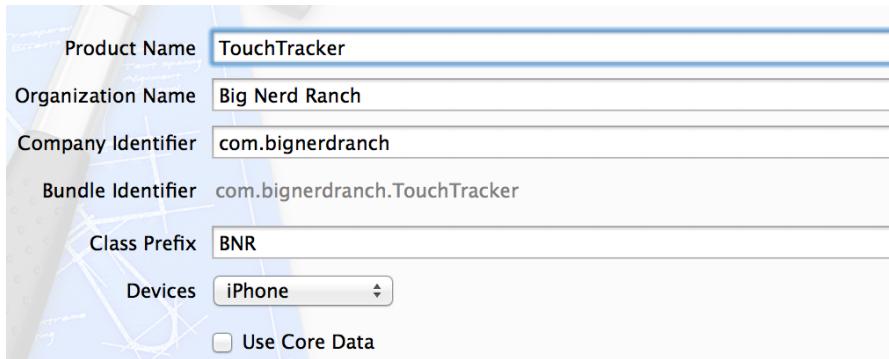
Sempre que um toque faz alguma coisa, como começar, se mover ou terminar, um *evento de toque* é adicionado a uma fila de eventos que o objeto **UIApplication** gerencia. Na prática, a fila raramente enche e os eventos são entregues imediatamente. A entrega destes eventos de toque envolve o envio de uma das mensagens de **UIResponder** à visão que é proprietária do toque. (Se os toques estiverem lentos, então, um de seus métodos está monopolizando a CPU e os eventos estão esperando na fila para serem entregues. O Chapter 14 mostrará como perceber estes problemas.)

E quanto a toques múltiplos? Se vários dedos fizerem a mesma coisa, exatamente ao mesmo tempo, à mesma visão, todos estes eventos de toque serão entregues de uma só vez. Cada objeto de toque – um para cada dedo – é incluído na **NSSet** que passa como um argumento nas mensagens de **UIResponder**. No entanto, a janela de oportunidade para o “mesmo exato momento” é bastante pequena. Portanto, em vez de uma mensagem do respondente para todos os toques, geralmente há várias mensagens do respondente com um ou mais toques.

## Criação do aplicativo TouchTracker

Vamos dar início ao seu aplicativo. No Xcode, crie um novo projeto Empty Application para iPhone e chame-o de TouchTracker. O prefixo de classe deve ser o mesmo que o de outros projetos, BNR (Figure 12.2).

Figure 12.2 Criação do TouchTracker



Primeiro, você vai precisar de um objeto de modelo que descreva uma linha. Crie uma nova subclasse de **NSObject** e nomeie-a como **BNRLine**. No **BNRLine.h**, declare duas propriedades **CGPoint**:

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

@interface BNRLine : NSObject

@property (nonatomic) CGPoint begin;
@property (nonatomic) CGPoint end;

@end
```

Em seguida, crie uma nova subclasse **NSObject** chamada **BNRDrawView**. No **BNRDrawView.h**, altere a superclasse para **UIView**.

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

@interface BNRDrawView : NSObject
@interface BNRDrawView : UIView

@end
```

Agora você precisa de um controlador de visão para gerenciar uma instância de **BNRDrawView** no TouchTracker. Crie uma nova subclasse **NSObject** chamada **BNRDrawViewController**. No **BNRDrawViewController.h**, altere a superclasse para **UIViewController**.

```
@interface BNRDrawViewController : NSObject
@interface BNRDrawViewController : UIViewController
```

No **BNRDrawViewController.m**, sobrescreva **loadView** para definir uma instância de **BNRDrawView** como **view** da **BNRDrawViewController**. Certifique-se de importar o arquivo de cabeçalho da **BNRDrawView** no topo desse arquivo.

```
#import "BNRDrawViewController.h"
#import "BNRDrawView.h"

@implementation BNRDrawViewController

- (void)loadView
{
    self.view = [[BNRDrawView alloc] initWithFrame:CGRectZero];
}

@end
```

No **BNRAppDelegate.m**, crie uma instância de **BNRDrawViewController** e defina-a como a **rootViewController** da janela. Não se esqueça de importar o arquivo de cabeçalho de **BNRDrawViewController** nesse arquivo.

```
#import "BNRAppDelegate.h"
#import "BNRDrawViewController.h"

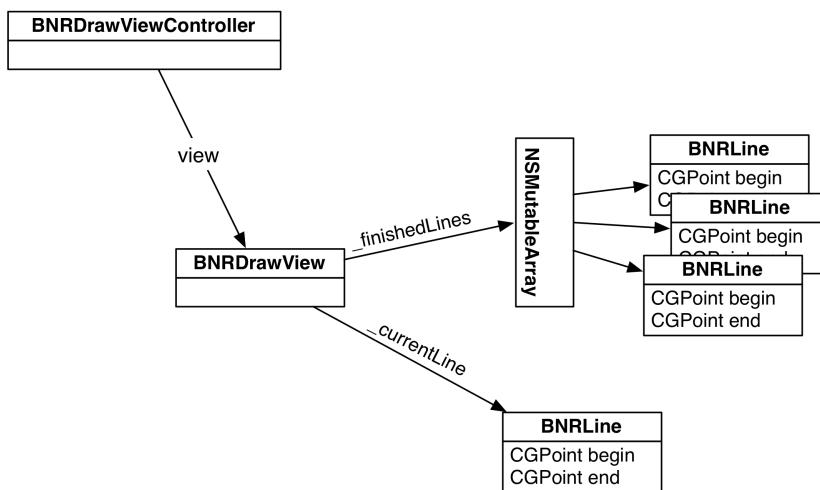
@implementation BNRAppDelegate

- (BOOL)application:(UIApplication *)application
    didFinishLaunchingWithOptions:(NSDictionary *)launchOptions
{
    self.window = [[UIWindow alloc] initWithFrame:[[UIScreen mainScreen] bounds]];
    // Override point for customization after application launch

    BNRDrawViewController *dvc = [[BNRDrawViewController alloc] init];
    self.window.rootViewController = dvc;

    self.window.backgroundColor = [UIColor whiteColor];
    [self.window makeKeyAndVisible];
    return YES;
}
```

Figure 12.3 Diagrama de objetos do TouchTracker



Os principais objetos que você acabou de definir para o TouchTracker são exibidos na Figure 12.3.

## Desenho com TouchDrawView

**BNRDrawView** controlará todas as linhas que já foram traçadas e qualquer uma sendo traçada atualmente. No **BNRDrawView.m**, crie duas variáveis de instância na extensão de classe que conterá as linhas em seus dois estados. Certifique-se de importar **BNRLLine.h** e implementar **initWithFrame:**.

```

#import "BNRDrawView.h"
#import "BNRLLine.h"

@interface BNRDrawView : NSObject

@property (nonatomic, strong) BNRLLine *currentLine;
@property (nonatomic, strong) NSMutableArray *finishedLines;

@end

@implementation BNRDrawView

- (instancetype)initWithFrame:(CGRect)r
{
    self = [super initWithFrame:r];
    if (self) {
        self.finishedLines = [[NSMutableArray alloc] init];
        self.backgroundColor = [UIColor grayColor];
    }
    return self;
}


```

Veremos como as linhas são criadas daqui a pouco, mas para verificar se o código de criação de linhas foi escrito corretamente, você precisa que a **BNRDrawView** seja capaz de traçar linhas.

No **BNRDrawView.m**, implemente **drawRect:** para traçar as linhas atuais e finalizadas.

```

- (void)strokeLine:(BNRLine *)line
{
    UIBezierPath *bp = [UIBezierPath bezierPath];
    bp.lineWidth = 10;
    bp.lineCapStyle = kCGLineCapRound;

    [bp moveToPoint:line.begin];
    [bp addLineToPoint:line.end];
    [bp stroke];
}

- (void)drawRect:(CGRect)rect
{
    // Draw finished lines in black
    [[UIColor blackColor] set];
    for (BNRLine *line in self.finishedLines) {
        [self strokeLine:line];
    }

    if (self.currentLine) {
        // If there is a line currently being drawn, do it in red
        [[UIColor redColor] set];
        [self strokeLine:self.currentLine];
    }
}

```

## Transformando toques em linhas

Uma linha é definida por dois pontos. A sua **BNRLine** armazena esses pontos como propriedades chamadas `begin` e `end`. Quando um toque for iniciado, você criará uma linha e colocará tanto `begin` quanto `end` no ponto em que o toque começou. Quando o toque se mover, você atualizará `end`. Quando o toque terminar, você terá a linha completa.

No `BNRDrawView.m`, implemente **touchesBegan:withEvent:** para criar uma nova linha.

```

- (void)touchesBegan:(NSSet *)touches
                  withEvent:(UIEvent *)event
{
    UITouch *t = [touches anyObject];

    // Get location of the touch in view's coordinate system
    CGPoint location = [t locationInView:self];

    self.currentLine = [[BNRLine alloc] init];
    self.currentLine.begin = location;
    self.currentLine.end = location;

    [self setNeedsDisplay];
}

```

Depois, no `BNRDrawView.m`, implemente **touchesMoved:withEvent:** para que ele atualize a propriedade `end` da `currentLine`.

```

- (void)touchesMoved:(NSSet *)touches
                  withEvent:(UIEvent *)event
{
    UITouch *t = [touches anyObject];
    CGPoint location = [t locationInView:self];

    self.currentLine.end = location;

    [self setNeedsDisplay];
}

```

Finalmente, no `BNRDrawView.m`, adicione a `currentLine` à `finishedLines` quando o toque terminar.

```
- (void)touchesEnded:(NSSet *)touches
              withEvent:(UIEvent *)event
{
    [self.finishedLines addObject:self.currentLine];
    self.currentLine = nil;
    [self setNeedsDisplay];
}
```

Compile e execute o aplicativo e depois trace algumas linhas na tela. Ao traçar as linhas, elas aparecerão em vermelho e, quando finalizadas, serão exibidas em preto.

## Lidando com toques múltiplos

Ao traçar as linhas, você deve ter percebido que ao colocar mais de um dedo na tela, nada acontece – isto porque você pode traçar apenas uma linha por vez. Vamos atualizar **BNRDrawView** para que você possa traçar tantas linhas quanto o número de dedos que couberem na tela.

Por padrão, uma visão aceitará apenas um toque por vez. Se um dedo já disparou **touchesBegan:withEvent:**, mas ainda não foi finalizado – e, portanto, não disparou **touchesEnded:withEvent:** – os toques posteriores serão ignorados. Neste contexto, “ignorar” quer dizer que **BNRDrawView** não receberá **touchesBegan:withEvent:** nem qualquer outra mensagem **UIResponder** relacionada aos toques extras.

No **BNRDrawView.m**, ative as instâncias de **BNRDrawView** para aceitar toques múltiplos.

```
- (instancetype)initWithFrame:(CGRect)r
{
    self = [super initWithFrame:r];

    if (self) {
        self.finishedLines = [[NSMutableArray alloc] init];
        self.backgroundColor = [UIColor grayColor];
        self.multipleTouchEnabled = YES;
    }

    return self;
}
```

Agora que **BNRDrawView** aceitará toques múltiplos, toda vez que um dedo tocar a tela, se mover ou for retirado da tela, a visão receberá a mensagem **UIResponder** adequada. No entanto, agora temos um problema: o seu código da **UIResponder** pressupõe que haverá apenas um toque ativo e uma linha sendo traçada de cada vez.

Observe, em primeiro lugar, que cada método para manuseio de toque que você já implementou envia a mensagem **anyObject** para a **NSSet** de **touches** que recebe. Em uma visão de toque único, sempre haverá apenas um objeto no conjunto, portanto, solicitar qualquer objeto sempre lhe dará o toque que disparou o evento. Em uma visão de toque múltiplo, esse conjunto poderia conter mais de um toque.

Observe, então, que há apenas uma propriedade (**currentLine**) ligada a uma linha em progresso. Obviamente, você precisará manter tantas linhas quanto o número de toques atualmente na tela. Embora você possa criar outras propriedades, como **currentLine1** e **currentLine2**, você teria muito trabalho para controlar qual variável de instância corresponde a qual toque.

Em vez da abordagem de múltiplas propriedades, você pode utilizar uma classe **NSTableDictionary** para ficar ligada a cada **BNRLine** em progresso. A chave para armazenar a linha no dicionário será derivada do objeto **UITouch** ao qual a linha corresponde. À medida que mais eventos de toque ocorrem, você pode utilizar o mesmo algoritmo para produzir a chave a partir da **UITouch** que disparou o evento e usá-la para procurar a **BNRLine** adequada no dicionário.

No **BNRDrawView.m**, adicione uma nova variável de instância para substituir a **currentLine** e instanciá-la no **initWithFrame:**.

```

@interface BNRDrawView ()

@property (nonatomic, strong) BNRLLine *currentLine;
@property (nonatomic, strong) NSMutableDictionary *linesInProgress;
@property (nonatomic, strong) NSMutableArray *finishedLines;

@end

@implementation BNRDrawView

- (instancetype)initWithFrame:(CGRect)r
{
    self = [super initWithFrame:r];
    if (self) {
        self.linesInProgress = [[NSMutableDictionary alloc] init];
        self.finishedLines = [[NSMutableArray alloc] init];
        self.backgroundColor = [UIColor grayColor];
        self.multipleTouchEnabled = YES;
    }
    return self;
}

```

Agora você precisa atualizar os métodos de **UIResponder** para adicionar ao dicionário as linhas que estão sendo traçadas atualmente. No `BNRDrawView.m`, atualize o código em `touchesBegan:withEvent:`:

```

- (void)touchesBegan:(NSSet *)touches
                  withEvent:(UIEvent *)event
{
    // Let's put in a log statement to see the order of events
    NSLog(@"%@", NSStringFromSelector(_cmd));

    for (UITouch *t in touches) {
        CGPoint location = [t locationInView:self];

        BNRLLine *line = [[BNRLLine alloc] init];
        line.begin = location;
        line.end = location;

        NSValue *key = [NSValue valueWithNonretainedObject:t];
        self.linesInProgress[key] = line;
    }

    UITouch *t = [touches anyObject];
    CGPoint location = [t locationInView:self];

    self.currentLine = [[BNRLLine alloc] init];
    self.currentLine.begin = location;
    self.currentLine.end = location;

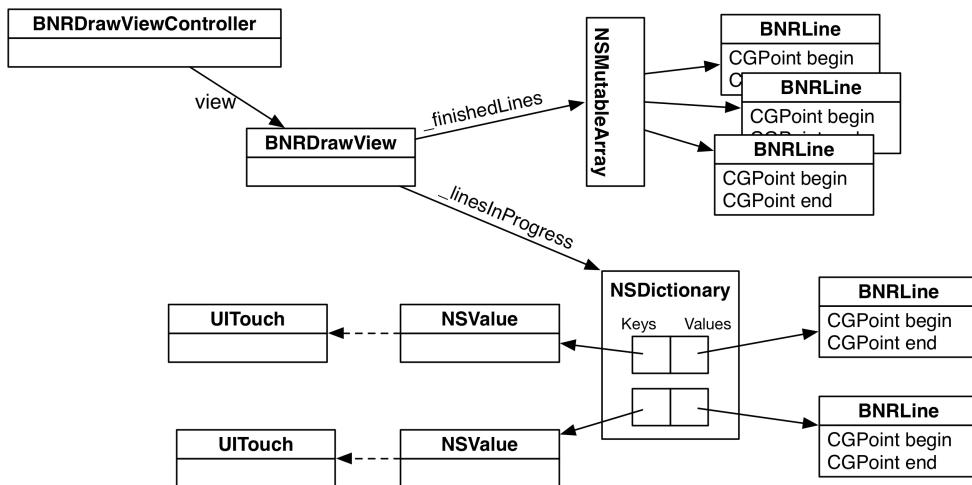
    [self setNeedsDisplay];
}

```

Primeiro, observe que você utiliza a enumeração rápida para executar o loop em todos os toques que foram iniciados, pois é possível que mais de um toque seja iniciado ao mesmo tempo. (Embora, normalmente, os toques iniciem em momentos diferentes e a `BNRDrawView` receberá várias mensagens `touchesBegan:withEvent:` contendo cada toque.)

Em seguida, observe o uso de `valueWithNonretainedObject:` para produzir a chave para armazenar a `BNRLLine`. Esse método cria uma instância de `NSValue` que contém o endereço do objeto `UITouch` que será associado a esta linha. Como uma `UITouch` é criada quando o toque é iniciado, atualizada ao longo do seu ciclo de vida e destruída quando o toque é finalizado, o endereço desse objeto será constante para cada mensagem de evento de toque.

Figure 12.4 Diagrama de objetos do TouchTracker multitoques



Atualize **touchesMoved:withEvent:** no **BNRDrawView.m** para que possa procurar a **BNRLine** certa.

```
- (void)touchesMoved:(NSSet *)touches
              withEvent:(UIEvent *)event
{
    // Let's put in a log statement to see the order of events
    NSLog(@"%@", NSStringFromSelector(_cmd));

    for (UITouch *t in touches) {
        NSValue *key = [NSValue valueWithNonretainedObject:t];
        BNRLine *line = self.linesInProgress[key];

        line.end = [t locationInView:self];
    }

    UITouch *t = [touches anyObject];
    CGPoint location = [t locationInView:self];

    self.currentLine.end = location;
    [self setNeedsDisplay];
}
```

Depois, atualize **touchesEnded:withEvent:** para mover qualquer linha finalizada para dentro do array de **\_finishedLines**.

```
- (void)touchesEnded:(NSSet *)touches
              withEvent:(UIEvent *)event
{
    // Let's put in a log statement to see the order of events
    NSLog(@"%@", NSStringFromSelector(_cmd));

    for (UITouch *t in touches) {
        NSValue *key = [NSValue valueWithNonretainedObject:t];
        BNRLine *line = self.linesInProgress[key];

        [self.finishedLines addObject:line];
        [self.linesInProgress removeObjectForKey:key];
    }

    [self.finishedLines addObject:self.currentLine];
    self.currentLine = nil;
    [self setNeedsDisplay];
}
```

Finalmente, atualize **drawRect:** para traçar cada linha na `_linesInProgress`.

```
// Draw finished lines in black
[[UIColor blackColor] set];
for (BNRLine *line in self.finishedLines) {
    [self strokeLine:line];
}

[[UIColor redColor] set];
for (NSValue *key in self.linesInProgress) {
    [self strokeLine:self.linesInProgress[key]];
}

if (self.currentLine) {
    // Draw line in progress in red
    [[UIColor redColor] set];
    [self strokeLine:self.currentLine];
}
}
```

Compile e execute o aplicativo e comece a traçar linhas com vários dedos. (Você pode simular vários dedos no simulador, mantendo a tela de opção pressionada enquanto arrasta.)

Você deve estar se perguntando: por que não utilizar a **UITouch** propriamente dita como chave? Por que se dar ao trabalho de criar uma **NSValue**? Os objetos usados como chave em uma **NSDictionary** devem estar de acordo com o protocolo **NSCopying**, que permite que os objetos sejam copiados por meio do envio da mensagem **copy**. As instâncias **UITouch** não estão de acordo com esse protocolo, pois não faz sentido que sejam copiadas. Desta forma, as instâncias **NSValue** contêm o endereço da **UITouch** para que instâncias **NSValue** iguais possam ser criadas posteriormente com a mesma **UITouch**.

Você também deve saber que, quando uma mensagem **UIResponder**, como **touchesMoved:withEvent:**, é enviada para uma visão, apenas os toques que se moveram estarão na **NSSet** de **touches**. Assim, é possível que três toques estejam em uma visão, mas apenas um toque estará dentro do conjunto de toques passados em um desses métodos se os outros dois não se moveram. Além disso, quando uma **UITouch** é iniciada em uma visão, todas as mensagens de evento de toque são enviadas para essa mesma visão durante o ciclo de vida do toque, mesmo que o toque saia da visão em que começou.

O último item que falta para os princípios básicos do TouchTracker é lidar com o que acontece quando um toque é cancelado. Um toque pode ser cancelado quando um aplicativo é interrompido pelo sistema operacional (por exemplo, uma ligação telefônica é recebida) quando um toque está na tela. Quando um toque é cancelado, qualquer estado que ele cria deve ser revertido. Neste caso, você deve remover qualquer linha em progresso.

No `BNRDrawView.m`, implemente **touchesCancelled:withEvent:**.

```
- (void)touchesCancelled:(NSSet *)touches
                  withEvent:(UIEvent *)event
{
    // Let's put in a log statement to see the order of events
    NSLog(@"%@", NSStringFromSelector(_cmd));

    for (UITouch *t in touches) {
        NSValue *key = [NSValue valueWithNonretainedObject:t];
        [self.linesInProgress removeObjectForKey:key];
    }

    [self setNeedsDisplay];
}
```

## Desafio de bronze: salvar e carregar

Salve as linhas quando o aplicativo for encerrado. Recarregue-as quando o aplicativo for reiniciado.

## Desafio de prata: cores

Faça com que o ângulo em que uma linha é traçada dite sua cor, depois de ser adicionada à `_finishedLines`.

## Desafio de ouro: círculos

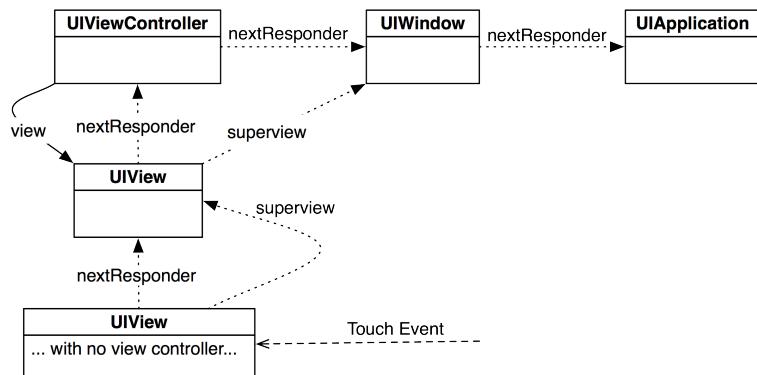
Use dois dedos para desenhar círculos. Tente fazer com que cada dedo represente um canto da caixa delimitadora ao redor do círculo. Você pode simular dois dedos no simulador, mantendo o botão Option pressionado. (Dica: isso é muito mais fácil se você controlar os toques que estão fazendo um círculo em um dicionário separado.)

## Para os mais curiosos: a cadeia de respondentes

No Chapter 7, falamos brevemente sobre a **UIResponder** e o primeiro respondente. Uma **UIResponder** pode receber eventos de toque. A **UIView** é um exemplo de uma subclasse **UIResponder**, mas há muitas outras, inclusive **UIViewController**, **UIApplication** e **UIWindow**. Você, provavelmente, está pensando: “Mas não é possível tocar uma **UIViewController**. Não é um objeto na tela.” Você está certo; você não pode enviar um evento de toque *diretamente* para uma **UIViewController**, mas controladores de visão podem receber eventos através de uma *cadeia de respondentes*.

Toda **UIResponder** tem um ponteiro chamado `nextResponder` e, juntos, estes objetos formam a cadeia de respondentes (Figure 12.5). Um evento de toque começa na visão que foi tocada. A `nextResponder` de uma visão é, normalmente, sua **UIViewController** (caso tenha uma) ou sua supervisão (caso não tenha). A `nextResponder` de um controlador de visão é, geralmente, a supervisão de sua visão. A maior supervisão é a janela. A `nextResponder` da janela é a instância singleton de **UIApplication**.

Figure 12.5 Cadeia de respondentes



Como é que uma **UIResponder** *não* lida com um evento? Ela encaminha a mesma mensagem para sua `nextResponder`. Isso é o que a implementação padrão de métodos como `touchesBegan:withEvent:` faz. Portanto, se um método não for sobreescrito, seu próximo respondente tentará lidar com o evento de toque. Se o aplicativo (o último objeto na cadeia de respondentes) não lidar com o evento, então, ele é descartado.

Você também pode enviar uma mensagem explicitamente para o próximo respondente. Digamos que há uma visão que controla toques, mas se ocorre um toque duplo, seu próximo respondente deve lidar com ele. O código teria a seguinte aparência:

```

- (void)touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event
{
    UITouch *touch = [touches anyObject];
    if (touch.tapCount == 2) {
        [[self nextResponder] touchesBegan:touches withEvent:event];
        return;
    }
    ...
    ... Go on to handle touches that are not double taps
}

```

## Para os mais curiosos: UIControl

A classe **UIControl** é a superclasse para várias classes em Cocoa Touch, inclusive **UIButton** e **UISlider**. Você viu como definir destinos e ações para estes controles. Agora, podemos examinar mais detalhadamente como **UIControl** sobrescreve os mesmos métodos de **UIResponder** que você implementou neste capítulo.

Na **UIControl**, cada *evento de controle* possível está associado a uma constante. Botões, por exemplo, geralmente enviam mensagens de ação no evento de controle `UIControlEventTouchUpInside`. Um destino registrado para este evento de controle receberá sua mensagem de ação somente se o usuário tocar no controle e, em seguida, tirar o dedo da tela, dentro do frame do controle. Basicamente, é uma batida.

Para um botão, no entanto, você pode ter ações em outros tipos de eventos. Por exemplo, você pode disparar um método se o usuário retirar o dedo *dentro ou fora* do frame. Atribuir o destino e a ação de forma programática teria a seguinte aparência:

```
[rButton addTarget:tempController
           action:@selector(resetTemperature:)
   forControlEvents:UIControlEventTouchUpInside | UIControlEventTouchUpInside];
```

Agora, considere como **UIControl** lida com `UIControlEventTouchUpInside`.

```
// Not the exact code. There is a bit more going on!
- (void)touchesEnded:(NSSet *)touches withEvent:(UIEvent *)event
{
    // Reference to the touch that is ending
    UITouch *touch = [touches anyObject];

    // Location of that point in this control's coordinate system
    CGPoint touchLocation = [touch locationInView:self];

    // Is that point still in my viewing bounds?
    if (CGRectContainsPoint(self.bounds, touchLocation))
    {
        // Send out action messages to all targets registered for this event!
        [self sendActionsForControlEvents:UIControlEventTouchUpInside];
    } else {
        // The touch ended outside the bounds, different control event
        [self sendActionsForControlEvents:UIControlEventTouchUpInside];
    }
}
```

Como estas ações são enviadas para o destino certo? Ao final das implementações de método de **UIResponder**, o controle envia a mensagem `sendActionsForControlEvents`: a si mesmo. Esse método analisa todos os pares destino-ação que o controle tem e, caso algum deles esteja registrado para o evento de controle passado como o argumento, tais destinos recebem uma mensagem de ação.

No entanto, um controle nunca envia a mensagem diretamente para seus destinos. Em vez disso, ele encaminha estas mensagens através do objeto **UIApplication**. Por que não fazer os controles enviarem mensagens de ação diretamente para os destinos? Controles também podem ter ações de destino `nil`. Se o destino de uma **UIControl** é `nil`, o **UIApplication** encontra o *primeiro respondente* de sua **UIWindow** e envia a mensagem de ação para ele.



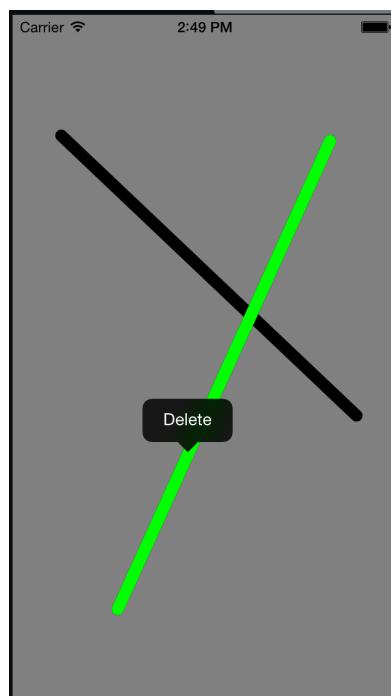
# 13

## UIGestureRecognizer e UIMenuController

No Chapter 12, você lidou com toques simples e determinou seus efeitos, implementando métodos da **UIResponder**. Às vezes, você quer detectar um padrão específico de toques que formam um gesto, como uma pinça ou o arraste de um dedo. Em vez de você mesmo escrever o código para detectar gestos comuns, você pode usar instâncias de **UIGestureRecognizer**.

Uma **UIGestureRecognizer** intercepta os toques que estão a caminho de serem tratados por uma visão. Quando reconhece um gesto específico, ela envia uma mensagem ao objeto de sua escolha. Há vários tipos de reconhecedores de gestos embutidos no SDK. Neste capítulo, você usará três deles para que os usuários do TouchTracker possam selecionar, mover e excluir linhas (Figure 13.1). Você verá também como usar outra classe interessante do iOS, a **UIMenuController**.

Figure 13.1 TouchTracker no final do capítulo



### Subclasses de UIGestureRecognizer

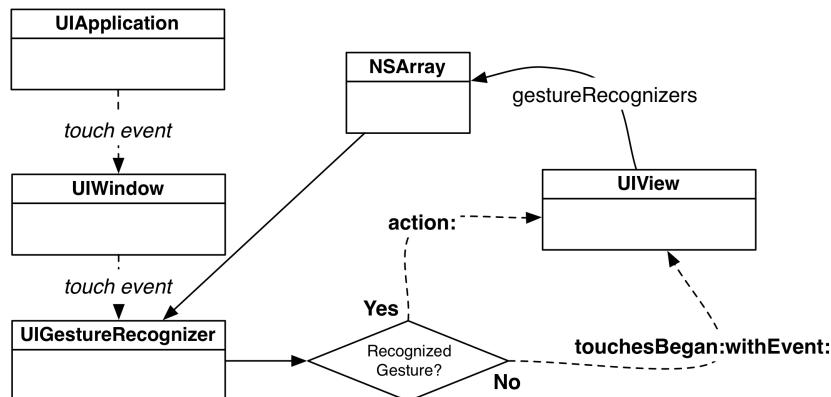
Você não vai instanciar a **UIGestureRecognizer** propriamente dita. Na verdade, há uma série de subclasses de **UIGestureRecognizer**, e cada uma é responsável por reconhecer um gesto específico.

Para usar uma instância de uma subclasse de **UIGestureRecognizer**, você fornece a ela um par destino-ação e o anexa a uma visão. Sempre que o reconhecedor de gestos reconhece seu gesto na visão, ele envia uma mensagem de ação ao seu destino. Todas as mensagens de ação da **UIGestureRecognizer** têm a mesma forma:

```
- (void)action:(UIGestureRecognizer *)gestureRecognizer;
```

Ao reconhecer um gesto, o reconhecedor de gestos intercepta os toques destinados à visão (Figure 13.2). Portanto, uma visão com reconhecedores de gestos pode não receber as mensagens típicas da **UIResponder**, como o **touchesBegan:withEvent:**.

Figure 13.2 Reconhecedores de gestos interceptam toques



## Detectção de toques curtos com UITapGestureRecognizer

A primeira subclasse de **UIGestureRecognizer** que você usará é **UITapGestureRecognizer**. Quando o usuário der dois toques curtos na tela, todas as linhas serão apagadas. Abra o TouchTracker.xcodeproj do Chapter 12.

No **BNRDrawView.m**, instancie uma **UITapGestureRecognizer** que requeira dois toques curtos para disparar no **initWithFrame:**.

```

- (instancetype)initWithFrame:(CGRect)r
{
    self = [super initWithFrame:r];
    if (self) {
        self.linesInProgress = [[NSMutableDictionary alloc] init];
        self.finishedLines = [[NSMutableArray alloc] init];
        self.backgroundColor = [UIColor grayColor];
        self.multipleTouchEnabled = YES;

        UITapGestureRecognizer *doubleTapRecognizer =
            [[UITapGestureRecognizer alloc] initWithTarget:self
                                              action:@selector(doubleTap:)];
        doubleTapRecognizer.numberOfTapsRequired = 2;

        [self addGestureRecognizer:doubleTapRecognizer];
    }
    return self;
}
  
```

Quando um toque duplo curto ocorre em uma instância de **BNRDrawView**, a mensagem **doubleTap:** é enviada para essa instância. Implemente esse método no **BNRDrawView.m**.

```

- (void)doubleTap:(UIGestureRecognizer *)gr
{
    NSLog(@"Recognized Double Tap");

    [self.linesInProgress removeAllObjects];
    [self.finishedLines removeAllObjects];
    [self setNeedsDisplay];
}
  
```

Observe que o argumento do método da ação para um reconhecedor de gestos é a instância de **UIGestureRecognizer** que enviou a mensagem. No caso de um toque duplo curto, você não precisa de nenhuma informação do reconhecedor, mas precisará de informações dos outros reconhecedores que você instalará posteriormente neste capítulo. Compile e execute o aplicativo, desenhe algumas linhas e toque duas vezes na tela para apagá-las.

Talvez você tenha percebido (principalmente no simulador) que, durante um toque duplo, o primeiro toque desenha um ponto vermelho pequeno. Esse ponto aparece porque **touchesBegan:withEvent:** é enviado para a **BNRDrawView** no primeiro toque, criando uma linha pequena. Verifique o console e você verá a seguinte sequência de eventos:

```
touchesBegan:withEvent:
Recognized Double Tap
touchesCancelled:withEvent:
```

Os reconhecedores de gestos trabalham inspecionando os eventos de toque para determinar se um gesto específico ocorreu. Antes de o gesto ser reconhecido, todas as mensagens da **UIResponder** são entregues para uma visão como normal. Embora o reconhecedor de gestos de toque reconheça quando um toque começa e termina dentro de uma pequena área em um curto período de tempo, a **UITapGestureRecognizer** ainda não pode alegar que o toque é um toque curto, e o **touchesBegan:withEvent:** é enviado para a visão. Quando o toque curto é finalmente reconhecido, o reconhecedor de gestos alega que o toque envolveu toque curto para si mesmo e nenhuma mensagem **UIResponder** é enviada mais para a visão desse toque específico. Para comunicar esse controle de toque para a visão, **touchesCancelled:withEvent:** é enviado à visão e a **NSSet** de touches contém essa instância de **UITouch**.

Para impedir temporariamente que esse ponto vermelho apareça, você pode dizer a uma **UIGestureRecognizer** para atrasar o envio do **touchesBegan:withEvent:** para sua visão, caso ainda seja possível o reconhecimento do gesto.

No **BNRDrawView.m**, modifique **initWithFrame:** para que ele faça exatamente isso.

```
UITapGestureRecognizer *doubleTapRecognizer =
    [[UITapGestureRecognizer alloc] initWithTarget:self
                                             action:@selector(doubleTap:)];
doubleTapRecognizer.numberOfTapsRequired = 2;
doubleTapRecognizer.delaysTouchesBegan = YES;
[self addGestureRecognizer:doubleTapRecognizer];
}
return self;
}
```

Compile e execute o aplicativo, desenhe algumas linhas e toque duas vezes na tela para apagá-las. Você não verá mais o ponto vermelho durante o toque duplo.

## Reconhecedores de múltiplos gestos

Vamos adicionar outro reconhecedor de gestos que permita que o usuário selecione uma linha. (Posteriormente, o usuário poderá excluir a linha selecionada.) Você instalará outra **UITapGestureRecognizer** na **BNRDrawView** que requer apenas um toque curto.

No **BNRDrawView.m**, modifique **initWithFrame:**.

```
[self addGestureRecognizer:doubleTapRecognizer];

UITapGestureRecognizer *tapRecognizer =
    [[UITapGestureRecognizer alloc] initWithTarget:self
                                             action:@selector(tap:)];
tapRecognizer.delaysTouchesBegan = YES;
[self addGestureRecognizer:tapRecognizer];
}
return self;
}
```

Agora, implemente **tap:** para registrar o toque curto no console do **BNRDrawView.m**.

```
- (void)tap:(UIGestureRecognizer *)gr
{
    NSLog(@"Recognized tap");
}
```

Compile e execute o aplicativo. Tocar uma vez registrará a mensagem apropriada no console. O único problema, contudo, é que o toque curto duplo disparará ambos, **tap:** e **doubleTap:**.

Nas situações em que você possui múltiplos reconhecedores de gestos, é comum disparar um reconhecedor de gestos quando você realmente quer que outro reconhecedor dê conta do trabalho. Nesses casos, você configura dependências entre reconhecedores que dizem: “Aguarde um momento antes de disparar, pois esse gesto pode ser meu!”.

No **initWithFrame:**, faça isso de forma que a `tapRecognizer` aguarde a `doubleTapRecognizer` falhar antes de assumir que um toque curto único não é exatamente o primeiro de um toque curto duplo.

```
UITapGestureRecognizer *tapRecognizer =  
    [[UITapGestureRecognizer alloc] initWithTarget:self  
                                            action:@selector(tap:)];  
  
tapRecognizer.delaysTouchesBegan = YES;  
[tapRecognizer requireGestureRecognizerToFail:doubleTapRecognizer];  
[self addGestureRecognizer:tapRecognizer];
```

Compile e execute o aplicativo. Um toque curto único leva agora uma pequena quantidade de tempo para disparar assim que o toque curto ocorre, porém o toque curto duplo não dispara mais a mensagem `tap:`.

Agora, vamos compilar a `BNRDrawView`, assim o usuário pode selecionar as linhas quando elas forem tocadas. Primeiramente, adicione uma propriedade que mantenha a linha selecionada para a extensão de classe no `BNRDrawView.m`.

```
@interface BNRDrawView()  
  
@property (nonatomic, strong) NSMutableDictionary *linesInProgress;  
@property (nonatomic, strong) NSMutableArray *finishedLines;  
  
@property (nonatomic, weak) BNRLLine *selectedLine;  
  
@end
```

(Observe que essa propriedade é `weak` (fraca): o array `finishedLines` manterá a referência forte para a linha e `selectedLine` será definida como `nil` caso a linha seja removida da `finishedLines` quando a tela for limpa.)

Agora, em **drawRect:**, adicione algum código à parte inferior do método para desenhar a linha selecionada em verde.

```
[[UIColor redColor] set];  
for (NSValue *key in self.linesInProgress) {  
    [self strokeLine:self.linesInProgress[key]];  
}  
  
if (self.selectedLine) {  
    [[UIColor greenColor] set];  
    [self strokeLine:self.selectedLine];  
}  
}
```

Implemente o **lineAtPoint:** no `BNRDrawView.m` para obter uma `BNRLLine` próxima ao ponto determinado.

```
- (BNRLLine *)lineAtPoint:(CGPoint)p  
{  
    // Find a line close to p  
    for (BNRLLine *l in self.finishedLines) {  
        CGPoint start = l.begin;  
        CGPoint end = l.end;  
  
        // Check a few points on the line  
        for (float t = 0.0; t <= 1.0; t += 0.05) {  
            float x = start.x + t * (end.x - start.x);  
            float y = start.y + t * (end.y - start.y);  
  
            // If the tapped point is within 20 points, let's return this line  
            if (hypot(x - p.x, y - p.y) < 20.0) {  
                return l;  
            }  
        }  
    }  
  
    // If nothing is close enough to the tapped point, then we did not select a line  
    return nil;  
}
```

(Existem maneiras melhores de implementar o `lineAtPoint:`, mas esta implementação simplista está boa para a proposta atual.)

O ponto de seu interesse, obviamente, é o ponto em que ocorreu o toque curto. Você pode obter facilmente essa informação. Toda `UIGestureRecognizer` tem um método `locationInView:`. Enviar essa mensagem ao reconhecedor de gestos dará a você as coordenadas em que o gesto ocorreu no sistema de coordenadas da visão que é passada como argumento.

No `BNRDrawView.m`, envie a mensagem `locationInView:` ao reconhecedor de gestos, passe o resultado para o `lineAtPoint:`, e defina a linha retornada como `selectedLine`.

```
- (void)tap:(UIGestureRecognizer *)gr
{
    NSLog(@"Recognized tap");

    CGPoint point = [gr locationInView:self];
    self.selectedLine = [self lineAtPoint:point];

    [self setNeedsDisplay];
}
```

Compile e execute o aplicativo. Desenhe algumas linhas e depois toque em uma delas. A linha tocada deve aparecer em verde, mas lembre-se de que demora alguns segundos para que o toque curto seja reconhecido como não sendo parte de um toque curto duplo.

## UIMenuController

Em seguida, você fará isso de forma que, quando o usuário selecionar uma linha, um menu apareça exatamente onde o usuário tocou, oferecendo a opção de excluir aquela linha. Há uma classe embutida chamada `UIMenuController` que produz esse tipo de menu (Figure 13.3). Um controlador de menu tem uma lista de objetos da `UIMenuItem` e é apresentado em uma visão existente. Cada item tem um título (o que aparece no menu) e uma ação (a mensagem que ele envia ao primeiro respondente da janela).

Figure 13.3 Uma `UIMenuController`



Há apenas uma `UIMenuController` por aplicativo. Quando quiser apresentar essa instância, você deve preenchê-la com itens de menu, fornecer a ela um retângulo no qual possa se apresentar e defini-la como visível.

Faça isso no método `tap:` do `BNRDrawView.m` se o usuário tiver tocado rapidamente em uma linha. Se o usuário tiver tocado rapidamente em algum lugar que não seja próximo de uma linha, a seleção da linha atualmente selecionada será removida, e o controlador de menu será ocultado.

```
- (void)tap:(UIGestureRecognizer *)gr
{
    NSLog(@"Recognized tap");

    CGPoint point = [gr locationInView:self];
    self.selectedLine = [self lineAtPoint:point];

    if (self.selectedLine) {

        // Make ourselves the target of menu item action messages
        [self becomeFirstResponder];

        // Grab the menu controller
        UIMenuController *menu = [UIMenuController sharedMenuController];

        // Create a new "Delete" UIMenuItem
        UIMenuItem *deleteItem = [[UIMenuItem alloc] initWithTitle:@"Delete"
                                                       action:@selector(deleteLine:)];
        menu.menuItems = @[deleteItem];

        // Tell the menu where it should come from and show it
        [menu setTargetRect:CGRectMake(point.x, point.y, 2, 2) inView:self];
        [menu setMenuVisible:YES animated:YES];
    } else {
        // Hide the menu if no line is selected
        [[UIMenuController sharedMenuController] setMenuVisible:NO animated:YES];
    }
    [self setNeedsDisplay];
}
```

Para que um controlador de menu apareça, uma visão que responda a, no mínimo, uma mensagem de ação nos itens de menu da **UIMenuController** deve ser o primeiro respondente da janela – é por isso que você envia a mensagem **becomeFirstResponder** para a **BNRDrawView** antes de configurar o controlador de menu.

Se tiver uma classe de visão personalizada que precisa se tornar o primeiro respondente, você deve sobrescrever **canBecomeFirstResponder**. No **BNRDrawView.m**, sobrescreva esse método para retornar YES.

```
- (BOOL)canBecomeFirstResponder
{
    return YES;
}
```

Você pode compilar e executar o aplicativo agora, mas quando você selecionar uma linha, o menu não aparecerá. Quando está sendo apresentado, o controlador de menu passa por cada item de menu e pergunta para o primeiro respondente se ele deve implementar a mensagem de ação para aquele item. Se o primeiro respondente não implementar aquele método, o controlador de menu não mostrará o item de menu associado. Se nenhum item de menu tiver mensagens de ação implementadas pelo primeiro respondente, o menu não será exibido de forma alguma.

Para fazer o item de menu Delete (e o próprio menu) aparecer, implemente **deleteLine:** no **BNRDrawView.m**.

```
- (void)deleteLine:(id)sender
{
    // Remove the selected line from the list of _finishedLines
    [self.finishedLines removeObject:self.selectedLine];

    // Redraw everything
    [self setNeedsDisplay];
}
```

Compile e execute o aplicativo. Desenhe uma linha, toque nela, e depois selecione Delete no item de menu.

## UILongPressGestureRecognizer

Vamos testar mais duas subclasses de **UIGestureRecognizer**: **UILongPressGestureRecognizer** e **UIPanGestureRecognizer**. Quando você mantém uma linha pressionada (um toque longo), essa linha é selecionada, sendo possível arrastá-la para qualquer lugar com o dedo (arraste de tela).

Nesta seção, vamos nos focar no reconhecimento de toque longo. No `BNRDrawView.m`, instancie uma `UILongPressGestureRecognizer` em `initWithFrame:` e adicione-a à `BNRDrawView`.

```
[self addGestureRecognizer:tapRecognizer];
UILongPressGestureRecognizer *pressRecognizer =
    [[UILongPressGestureRecognizer alloc] initWithTarget:self
                                                action:@selector(longPress:)];
[self addGestureRecognizer:pressRecognizer];
```

Agora, quando o usuário mantiver pressionada a `BNRDrawView`, a mensagem `longPress:` será enviada a ela. Por padrão, um toque deve durar 0,5 segundo para se tornar um toque longo, porém, você pode alterar a `minimumPressDuration` do reconhecedor de gestos, se quiser.

Até aqui, você trabalhou com gestos de toque. Um toque curto é um gesto simples. Quando ele é reconhecido, o gesto já terminou, e a mensagem de ação já foi entregue. O toque longo, por outro lado, é um gesto que ocorre ao longo do tempo e é definido por três eventos separados.

Por exemplo, quando o usuário toca em uma visão, o reconhecedor de toque longo nota um *possível* toque longo, mas deve esperar para ver se o toque vai durar o suficiente para ser considerado um gesto de toque longo.

Quando o usuário segura o toque pelo tempo suficiente, o toque longo é reconhecido e o gesto *começa*. Quando o usuário remove o dedo, o gesto *termina*.

Cada um desses eventos causa uma alteração na propriedade `state` do reconhecedor de gestos. Por exemplo, a `state` do reconhecedor de toque longo descrito acima seria `UIGestureRecognizerStatePossible`, e então `UIGestureRecognizerStateBegan`, e finalmente `UIGestureRecognizerStateChanged`.

Quando um reconhecedor de gestos passa para qualquer estado que não seja o de possível toque longo, ele envia sua mensagem de ação ao seu destino. Isso significa que o destino do reconhecedor de toque longo recebe a mesma mensagem quando o toque longo inicia ou termina. O estado do reconhecedor de gestos permite que o destino determine porque ele recebeu a mensagem de ação, e tome a providência necessária.

Aqui está o plano para implementar nosso método de ação `longPress:`. Quando a visão receber o `longPress:` e o toque longo começar, você selecionará a linha mais próxima de onde o gesto ocorreu. Isso permite que o usuário selecione uma linha enquanto mantém o dedo na tela (o que será importante na próxima seção, quando você implementar o arraste de tela). Quando a visão receber o `longPress:` e o toque longo terminar, você desmarcará a linha.

No `BNRDrawView.m`, implemente `longPress:`.

```
- (void)longPress:(UIGestureRecognizer *)gr
{
    if (gr.state == UIGestureRecognizerStateBegan) {
        CGPoint point = [gr locationInView:self];
        self.selectedLine = [self lineAtPoint:point];

        if (self.selectedLine) {
            [self.linesInProgress removeAllObjects];
        }
    } else if (gr.state == UIGestureRecognizerStateChanged) {
        self.selectedLine = nil;
    }
    [self setNeedsDisplay];
}
```

Compile e execute o aplicativo. Desenhe uma linha, pressione-a e segure; a linha ficará verde e será selecionada, e ficará assim até você liberá-la.

## UIPanGestureRecognizer e reconhecedores simultâneos

Quando uma linha é selecionada durante um toque longo, você deseja que o usuário consiga movimentar essa linha pela tela, arrastando-a com o dedo. Você precisa então de um reconhecedor de gestos para o dedo que se move pela tela. Esse gesto chama-se *arraste de tela*, e a subclasse do seu reconhecedor de gestos é a `UIPanGestureRecognizer`.

Geralmente, um reconhecedor de gestos não compartilha os toques que intercepta. Depois de reconhecer o gesto, ele “consome” aquele toque, e nenhum outro reconhecedor tem a chance de tratá-lo. No seu caso, isso é ruim: o gesto de arraste de tela que você quer reconhecer ocorre dentro de um gesto de toque longo. Você precisa que o reconhecedor de toque longo e o reconhecedor de arraste de tela consigam reconhecer seus respectivos gestos simultaneamente. Vamos ver como fazer isso.

Primeiro, na extensão de classe do `BNRDrawView.m`, declare que `BNRDrawView` está em conformidade com o protocolo `UIGestureRecognizerDelegate`. Em seguida, declare uma `UIPanGestureRecognizer` como propriedade, assim você tem acesso a ela em todos os seus métodos.

```
@interface BNRDrawView () <UIGestureRecognizerDelegate>

@property (nonatomic, strong) UIPanGestureRecognizer *moveRecognizer;
@property (nonatomic, strong) NSMutableDictionary *linesInProgress;
@property (nonatomic, strong) NSMutableArray *finishedLines;

@property (nonatomic, weak) BNRLLine *selectedLine;

@end
```

No `BNRDrawView.m`, adicione código ao `initWithFrame:` para instanciar uma `UIPanGestureRecognizer`, defina duas de suas propriedades e as associe à `BNRDrawView`.

```
[self addGestureRecognizer:pressRecognizer];

self.moveRecognizer = [[UIPanGestureRecognizer alloc] initWithTarget:self
                                                       action:@selector(moveLine:)];
self.moveRecognizer.delegate = self;
self.moveRecognizer.cancelsTouchesInView = NO;
[self addGestureRecognizer:self.moveRecognizer];
```

Existem diversos métodos no protocolo `UIGestureRecognizerDelegate`, mas você está interessado apenas em um deles: `gestureRecognizer:shouldRecognizeSimultaneouslyWithGestureRecognizer:`. Um reconhecedor de gestos envia essa mensagem a sua delegate quando ele reconhece seu respectivo gesto, mas percebe que outro reconhecedor de gestos também reconheceu o mesmo gesto. Se esse método retornar YES, o reconhecedor compartilhará seus toques com outros reconhecedores de gesto.

No `BNRDrawView.m`, retorne YES quando `_moveRecognizer` enviar a mensagem ao seu respectivo delegate.

```
- (BOOL)gestureRecognizer:(UIGestureRecognizer *)gestureRecognizer
    shouldRecognizeSimultaneouslyWithGestureRecognizer:(UIGestureRecognizer *)other
{
    if (gestureRecognizer == self.moveRecognizer) {
        return YES;
    }
    return NO;
}
```

Agora, quando o usuário iniciar um toque longo, `UIPanGestureRecognizer` também conseguirá acompanhar esse dedo. Quando o dedo começar a se mover, o reconhecedor de arraste de tela passará para o estado de início. Se esses dois reconhecedores não pudesse trabalhar simultaneamente, o reconhecer de toque longo iniciaria, e o reconhecedor de arraste de tela nunca passaria para o estado de início nem enviaria sua mensagem de ação ao seu destino.

Além dos estados que você já viu, um reconhecedor de arraste de tela suporta o estado *alterado*. Quando o dedo começa a se mover, o reconhecer de arraste de tela entra no estado de início e envia uma mensagem ao seu destino. Enquanto o dedo se move em torno da tela, o reconhecedor passa para o estado de alterado, e envia sua mensagem de ação ao seu destino várias vezes. Finalmente, quando o dedo deixa de tocar a tela, o reconhecedor passa para o estado de término, e a mensagem final é entregue ao destino.

Agora, precisamos implementar o método `moveLine:`: que o reconhecedor de arraste de tela envia ao seu destino. Nessa implementação, você enviará a mensagem `translationInView:` ao reconhecedor de arraste de tela. Esse método da `UIPanGestureRecognizer` retorna até onde o arraste se moveu na forma de um `CGPoint` no sistema de coordenadas da visão, passado como argumento. Quando começa o gesto de arraste, essa propriedade é definida para o ponto zero (em que x e y são iguais a zero). Conforme o arraste se move, esse valor é atualizado

– se o arraste for muito para a direita, ele terá um valor de x alto; se o arraste retornar para onde começou, sua tradução volta para o ponto zero.

No BNRDrawView.m, implemente `moveLine:`. Observe que, como você enviará ao reconhecedor de gestos um método a partir da classe **UIPanGestureRecognizer**, o parâmetro desse método deve ser um ponteiro para uma instância de **UIPanGestureRecognizer** e não de **UIGestureRecognizer**.

```
- (void)moveLine:(UIPanGestureRecognizer *)gr
{
    // If we have not selected a line, we do not do anything here
    if (!self.selectedLine)
        return;

    // When the pan recognizer changes its position...
    if (gr.state == UIGestureRecognizerStateChanged) {
        // How far has the pan moved?
        CGPoint translation = [gr translationInView:self];

        // Add the translation to the current beginning and end points of the line
        CGPoint begin = self.selectedLine.begin;
        CGPoint end = self.selectedLine.end;
        begin.x += translation.x;
        begin.y += translation.y;
        end.x += translation.x;
        end.y += translation.y;

        // Set the new beginning and end points of the line
        self.selectedLine.begin = begin;
        self.selectedLine.end = end;

        // Redraw the screen
        [self setNeedsDisplay];
    }
}
```

Compile e execute o aplicativo. Toque e mantenha o dedo sobre uma linha e comece a arrastar; você verá imediatamente que a linha e o seu dedo estão fora de sincronia. Isso faz sentido porque você está adicionando a tradução atual repetidas vezes aos pontos finais originais da linha. O que você realmente precisa é que o reconhecedor de gestos reporte a mudança na tradução desde a última vez que esse método foi chamado. E felizmente, você pode fazer isso. Você pode definir a tradução de um reconhecedor de arraste de tela de volta para o ponto zero toda vez que ele reportar uma alteração. Dessa forma, na próxima vez em que ele reportar uma alteração, terá a tradução desde o último evento.

Próximo à parte inferior do `moveLine:` no BNRDrawView.m, adicione a linha de código a seguir.

```
[self setNeedsDisplay];
[gr setTranslation:CGPointZero inView:self];
}
```

Compile e execute o aplicativo, e movimente uma linha pela tela. Funciona direitinho!

Antes de prosseguir, vamos dar uma olhada em uma propriedade que você definiu no reconhecedor de arraste de tela: `cancelsTouchesInView`. Toda **UIGestureRecognizer** tem essa propriedade e, por padrão, essa propriedade é YES. Isso significa que o reconhecedor de gestos consome qualquer toque que reconhece para que a visão não tenha a chance de tratar o toque pelos métodos tradicionais da **UIResponder** como, por exemplo, o `touchesBegan:withEvent:`.

Geralmente, é isso que você quer, mas nem sempre. Neste caso, o gesto que o reconhecedor de arraste de tela reconhece é o mesmo tipo de toque que a visão processa para desenhar linhas usando os métodos da **UIResponder**. Se o reconhecedor de gestos consumir esses toques, os usuários não conseguirão desenhar linhas.

Quando você define `cancelsTouchesInView` para NO, os toques reconhecidos pelo reconhecedor de gestos também são entregues à visão pelos métodos da **UIResponder**. Isso permite que o reconhecedor e os métodos da **UIResponder** da visão processem os mesmos toques. Se você estiver curioso, assinale a linha que define `cancelsTouchesInView` para NO e compile e execute novamente para ver os efeitos.

## Para os mais curiosos: UIMenuController e UIResponderStandardEditActions

**UIMenuController** é normalmente responsável por mostrar ao usuário um menu “edit” quando é exibida; é como um campo de texto ou uma visão de texto quando você pressiona e segura. Portanto, um controlador de menu não modificado (um para o qual você não define os itens de menu) já tem itens de menu padrão que são apresentados por ele, como Cut, Copy e outras opções que já conhecemos bem. Cada item tem uma mensagem de ação encadeada. Por exemplo, **cut:** é enviado à visão que está apresentando o controlador de menu quando o item de menu Cut é tocado.

Todas as instâncias de **UIResponder** implementam esses métodos mas, por padrão, esses métodos não fazem nada. Subclasses como a **UITextField** sobrescrevem esses métodos para fazer algo apropriado para o seu contexto, como cortar o texto atualmente selecionado. Os métodos são todos declarados no protocolo **UIResponderStandardEditActions**.

Se você sobrescrever um método de **UIResponderStandardEditActions** em uma visão, seu item de menu aparecerá automaticamente em qualquer menu que você exibir naquela visão. Isso funciona porque o controlador de menu envia a mensagem **canPerformAction:withSender:** à sua visão, que retorna YES ou NO, caso implemente ou não esse método.

Se você quer implementar um desses métodos, mas *não* quer que ele apareça no menu, pode sobrescrever o **canPerformAction:withSender:** para retornar NO.

```
- (BOOL)canPerformAction:(SEL)action withSender:(id)sender
{
    if (action == @selector(copy:))
        return NO;

    // The superclass's implementation will return YES if the method is in the .m file
    return [super canPerformAction:action withSender:sender];
}
```

## Para os mais curiosos: mais sobre UIGestureRecognizer

Falamos apenas superficialmente de **UIGestureRecognizer**; há mais subclasses, mais propriedades e mais métodos delegates, e você pode até criar seus próprios reconhecedores. Esta seção dará a você uma ideia do que a **UIGestureRecognizer** é capaz de fazer, e depois você pode consultar a documentação da **UIGestureRecognizer** para saber ainda mais.

Quando um reconhecedor de gestos está em uma visão, ele está tratando para você de todos os métodos da **UIResponder**, como o **touchesBegan:withEvent:**. Os reconhecedores de gestos são bem gananciosos, e geralmente não deixam que a visão receba eventos de toques, ou pelo menos adiam a entrega desses eventos. Você pode definir propriedades no reconhecedor, como **delaysTouchesBegan**, **delaysTouchesEnded** e **cancelsTouchesInView**, para alterar esse comportamento. Se precisar de um controle ainda maior do que essa abordagem tudo-ou-nada, você pode implementar métodos delegates para o reconhecedor.

Às vezes, você pode ter dois reconhecedores de gestos buscando gestos muito semelhantes. Você pode encadear os reconhecedores juntos de modo que um seja obrigado a falhar para que o outro comece a usar o método **requireGestureRecognizerToFail:**.

Uma coisa que você precisa compreender para dominar os reconhecedores de gestos é como eles interpretam seus respectivos estados. No geral, há sete estados em que o reconhecedor pode entrar:

- **UIGestureRecognizerStatePossible**
- **UIGestureRecognizerStateBegan**
- **UIGestureRecognizerStateChanged**
- **UIGestureRecognizerStateEnded**
- **UIGestureRecognizerStateFailed**
- **UIGestureRecognizerStateCancelled**
- **UIGestureRecognizerStateRecognized**

Na maior parte do tempo, o reconhecedor fica no estado de possível gesto. Quando o reconhecedor reconhece seu respectivo gesto, ele entra no estado de início. Se o gesto for algo que possa se estender, como o arraste de tela, ele entrará e ficará no estado de alterado até que o gesto termine. Quando qualquer uma de suas

propriedades muda, ele envia outra mensagem ao seu destino. Quando o gesto termina (geralmente quando o usuário levanta o dedo), ele entra no estado de término.

Nem todos os reconhecedores iniciam, mudam e terminam. Para os reconhecedores de gestos que detectam gestos discretos como um toque curto, a única coisa que você vai conseguir ver é o estado de reconhecido (que tem o mesmo valor do estado de término).

Finalmente, um reconhecedor pode ser cancelado (por uma chamada recebida, por exemplo) ou falhar (porque nem a contorção máxima dos dedos possibilitaria que o gesto fosse feito da posição que os dedos se encontram). Quando ocorre a transição por esses estados, a mensagem de ação do reconhecedor é enviada, e a propriedade de estado pode ser verificada para se descobrir o por quê.

Os três reconhecedores embutidos que você não implementou neste capítulo são **UIPinchGestureRecognizer**, **UISwipeGestureRecognizer** e **UIRotationGestureRecognizer**. Cada um deles tem propriedades que permitem ajustar detalhadamente seu comportamento. A documentação mostrará como fazer isso.

Finalmente, se houver um gesto que você quer reconhecer que não seja implementado pelas subclasses embutidas de **UIGestureRecognizer**, você pode criar suas próprias subclasses de **UIGestureRecognizer**. Esse é um trabalho complexo e fora do escopo deste livro. Você pode ler o Subclassing Notes na documentação da **UIGestureRecognizer** para aprender o que é preciso para fazer isso.

## Desafio de prata: linhas misteriosas

Há um bug no aplicativo: se tocar em uma linha e depois começar a desenhar uma nova enquanto o menu estiver visível, você arrastará a linha selecionada e desenhárá a nova linha ao mesmo tempo. Corrija esse bug.

## Desafio de ouro: velocidade e tamanho

Pegue carona com o reconhecedor de arraste de tela para registrar a velocidade do arraste enquanto estiver desenhando uma linha. Ajuste a espessura da linha que estiver sendo desenhada, com base nessa velocidade. Não pressuponha nada em relação a quanto baixo ou alto o valor da velocidade do reconhecedor de arraste de tela pode ser. (Ou seja, antes de tudo, registre uma variedade de velocidades no console.)

## Super desafio de ouro: cores

Faça com que um arraste de três dedos para cima faça aparecer um painel com algumas cores. A seleção de uma dessas cores deve fazer com que qualquer linha que você desenhe depois disso seja da cor selecionada. Nenhuma linha extra deve ser desenhada para fazer aparecer esse painel – ou, no mínimo, quaisquer linhas que sejam desenhadas devem ser imediatamente excluídas quando o aplicativo perceber que está trabalhando com um arraste de três dedos.



# 14

## Ferramentas de depuração

No Chapter 7, você aprendeu sobre como usar o depurador para encontrar e corrigir problemas no código. Agora, analisaremos outras ferramentas disponíveis para programadores de iOS e a forma como você pode integrá-las ao desenvolvimento de seus aplicativos.

### Medidores

O Xcode 5 apresentou os *medidores de depuração*, que oferecem informações gerais sobre a utilização de CPU e memória do seu aplicativo.

Abra o projeto TouchTracker e execute-o, de preferência em um dispositivo iOS provisionado em vez do iOS Simulator. No navegador, selecione a guia para abrir o navegador de depuração.

Figure 14.1 Medidores

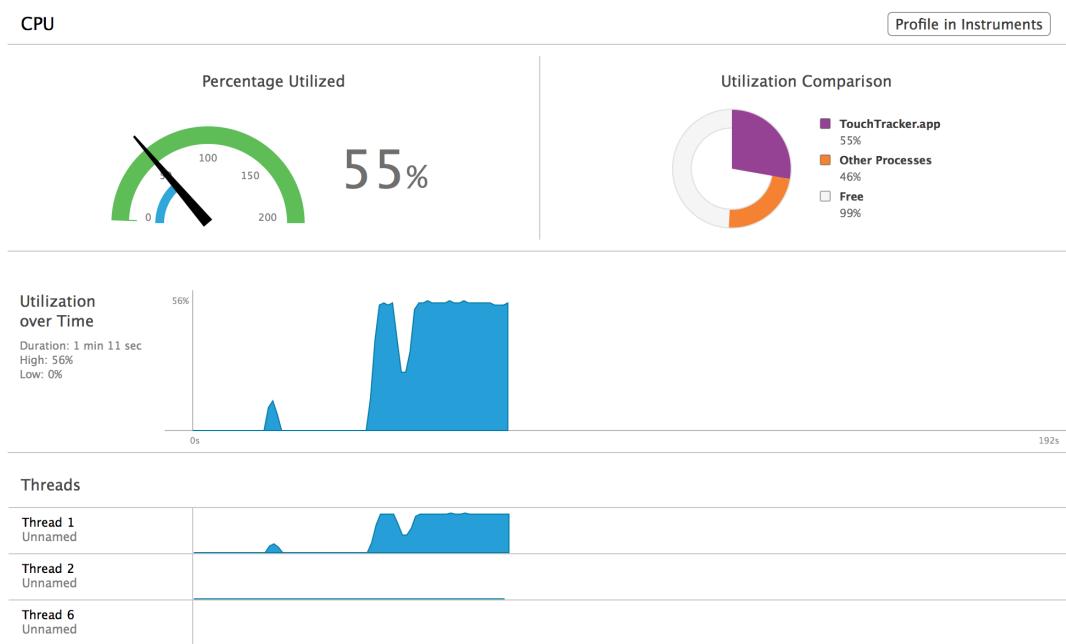


Enquanto o aplicativo está sendo executado (não pausado ou travado), o navegador de depuração mostra os medidores da CPU e da memória (Figure 14.1). Cada uma delas mostra um gráfico em tempo real da utilização de recursos ao longo do tempo, bem como dados numéricos que descrevem essa utilização atual de recursos.

Observação importante: esses medidores funcionam com base no hardware que está sendo executado em seu aplicativo. Seu Mac tem muito mais RAM disponível e provavelmente mais núcleos de CPU que os dispositivos iOS, portanto, se você executar seu aplicativo no iOS Simulator, sua utilização de CPU e memória parecerá estar muito baixa.

Clique em CPU Debug Gauge. O CPU Report será exibido no painel Editor (Figure 14.2).

Figure 14.2 Relatório da CPU



O relatório contém quatro seções básicas:

#### Percentage Utilized

mostra sua utilização de CPU em relação ao número de núcleos de CPU que seu dispositivo possui. Por exemplo, dispositivos com dois núcleos mostraram utilização de CPU de 200%. Quando seu aplicativo estiver ocioso, será exibido 0%.

#### Utilization Comparison

permite que você veja a utilização de CPU do seu aplicativo à medida que ele impacta o resto do sistema. Em determinado momento, seu aplicativo não será a única causa de atividade no dispositivo. Alguns aplicativos podem estar sendo executados em segundo plano, colocando sua própria pressão no sistema. Se seu aplicativo ficar lento, mas não estiver usando muito a CPU sozinho, esse pode ser o motivo.

#### Utilization over Time

exibe em gráfico a utilização de CPU do seu aplicativo e mostra por quanto tempo o aplicativo está sendo executado, além dos valores de utilização em pico e queda ao longo da execução atual.

#### Threads

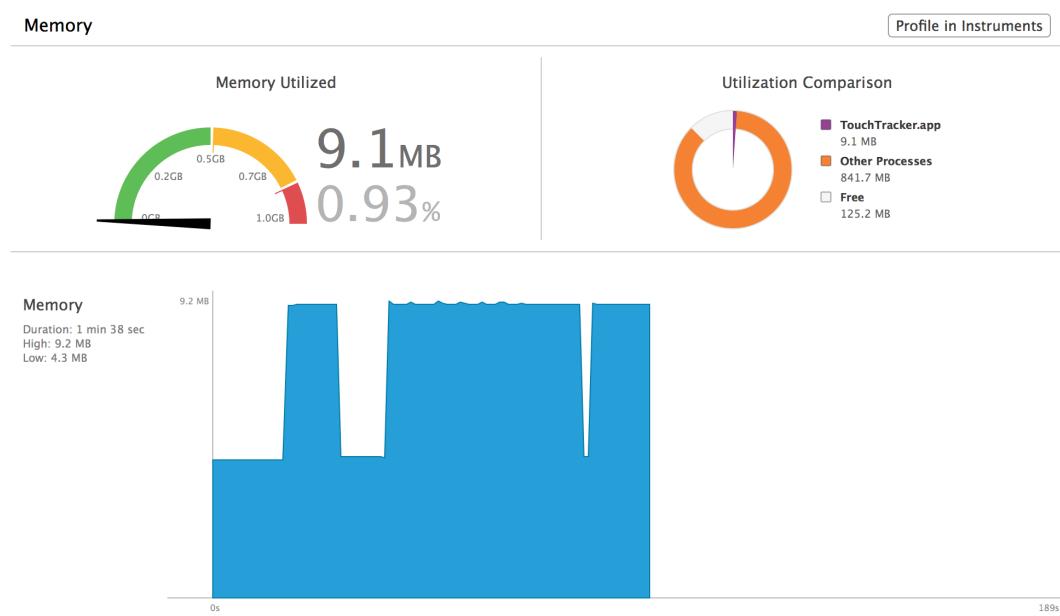
mostra em detalhes o gráfico de Utilization over Time por thread. Multithreading (multisegmentação) está fora do escopo deste livro, porém essas informações serão úteis à medida que você prossegue em sua carreira e treinamento em desenvolvimento para iOS.

Para deixar o gráfico um pouco menos chato, comece desenhando uma linha, mas continue movimentando seu dedo sem deixar a linha ficar travada em um só lugar. Isso causará um aumento prolongado na utilização da CPU.

Por quê? Cada ponto na tela em que seu dedo se movimenta causa uma volta no loop de execução do aplicativo, começando com uma mensagem `touchesMoved:withEvent:`, que por sua vez faz o `drawRect:` ser enviado para sua instância `BNRDrawView`. Quanto mais você trabalhar nesses métodos, mais utilização de CPU seu aplicativo exigirá enquanto as linhas estiverem sendo desenhadas.

Em seguida, no navegador de depuração, clique no Memory Debug Gauge para apresentar o Memory Report (Figure 14.3).

Figure 14.3 Relatório da memória



Assim como o CPU Report, o Memory Report é dividido em seções de fácil leitura. Não fique surpreso se seu gráfico de Memory (a seção inferior) parecer estar a 100%; esse gráfico faz o dimensionamento de forma que sua utilização da memória de pico represente 100%, visualmente.

Trata-se de um objetivo geral de desenvolvimento de software para qualquer plataforma, para manter a utilização da CPU e da memória o mais baixa possível, maximizando o desempenho do aplicativo para o usuário. É recomendável adquirir o hábito de verificar esses medidores e relatórios com antecedência e frequência em seus projetos, assim será mais provável que você perceba quando uma mudança que você fez no código resultou em uma alteração inesperada na utilização de recursos de seu aplicativo.

## Instrumentos

Os medidores e relatórios oferecem acesso fácil e rápido a uma compreensão de alto nível da utilização de recursos de seu aplicativo. Se sua utilização de CPU ou memória parecer mais alta do que deveria, ou se seu aplicativo ficar meio lento, você precisará de informações além daquelas que os medidores e relatórios oferecem.

O Instruments é um aplicativo que acompanha o Xcode, que você pode usar para monitorar seus aplicativos durante a execução e coletar estatísticas detalhadas sobre o desempenho do seu aplicativo. O Instruments é composto por diversos plug-ins que permitem que você inspecione alocações de objetos, utilização de CPU por função ou método, E/S de arquivo, E/S de rede e muito mais. Cada plug-in é conhecido como um Instrument. Juntos, eles ajudam você a acompanhar os déficits de desempenho de seu aplicativo.

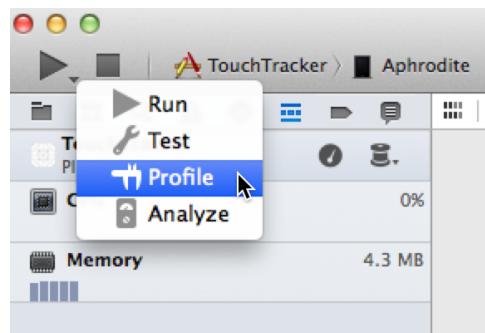
Ao usar o Instruments para monitorar seu aplicativo, você está *criando perfis* do aplicativo. Como os medidores de depuração, você pode criar perfis do aplicativo no simulador, mas obterá dados mais precisos em um dispositivo.

## Instrumento Allocations

O instrumento Allocations informa você sobre cada objeto criado e quanta memória ele ocupa.

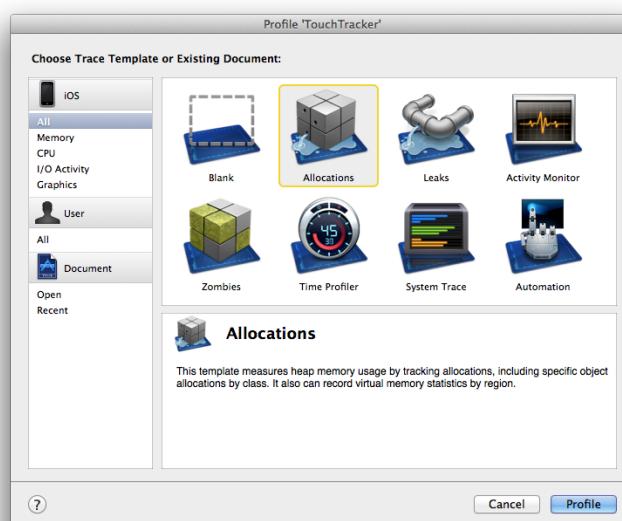
Para criar perfis de um aplicativo, clique e mantenha pressionado o botão Run no canto superior esquerdo da área de trabalho. No menu pop-up que é exibido, selecione Profile (Figure 14.4).

Figure 14.4 Criação de perfil de aplicativo



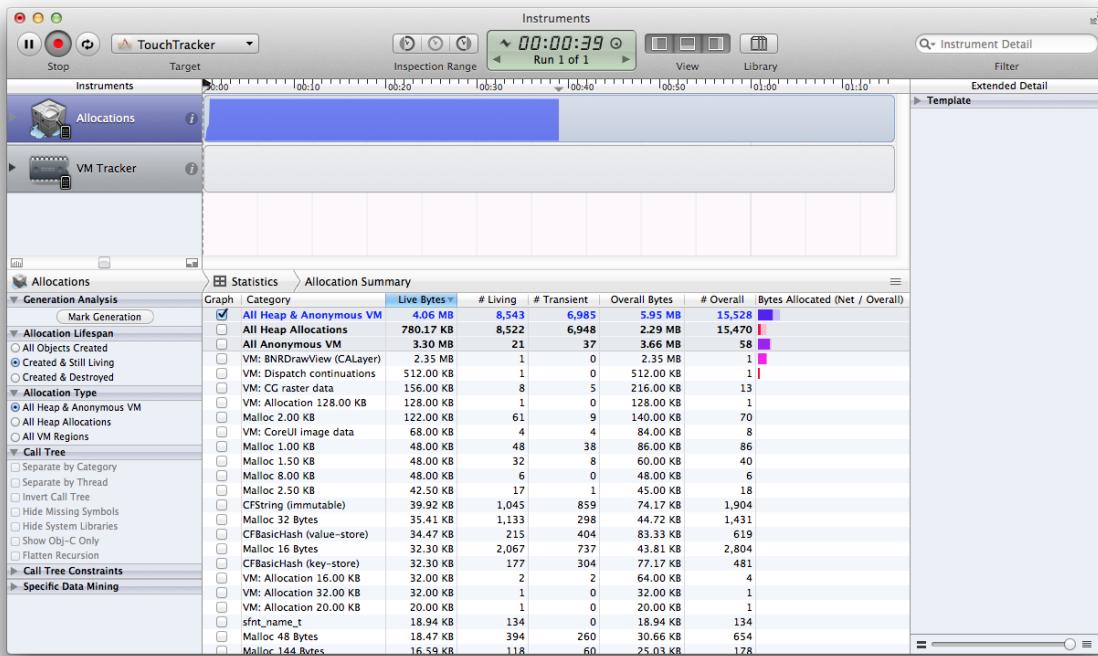
O Instruments será inicializado e perguntará que template de instrumento deverá usar. Observe que existem mais oito opções que você pode explorar se rolar a tela. Escolha Allocations e clique em Profile (Figure 14.5).

Figure 14.5 Escolha de um instrumento



O TouchTracker será inicializado e uma janela será aberta no Instruments (Figure 14.6). A interface pode parecer confusa no início, porém, assim como a janela da área de trabalho do Xcode, vai se tornar familiar com o tempo e a utilização. Primeiro, verifique se você pode ver tudo, ativando todas as áreas da janela. No controle View, na parte superior da janela, clique nos três botões para revelar as três áreas principais. A janela deve ser semelhante à da Figure 14.6.

Figure 14.6 Instrumento Allocations



Essa tabela exibe toda a alocação de memória do aplicativo. Há vários objetos aqui, mas vamos analisar aqueles que o seu código é responsável por criar. Primeiro, trace algumas linhas no TouchTracker. Em seguida, digite **BNRLine** na caixa de pesquisa **Instrument Detail**, no canto superior direito da janela.

Isso filtrará a lista de objetos na tabela **Object Summary** para que ela mostre apenas as instâncias de **BNRLine** (Figure 14.7).

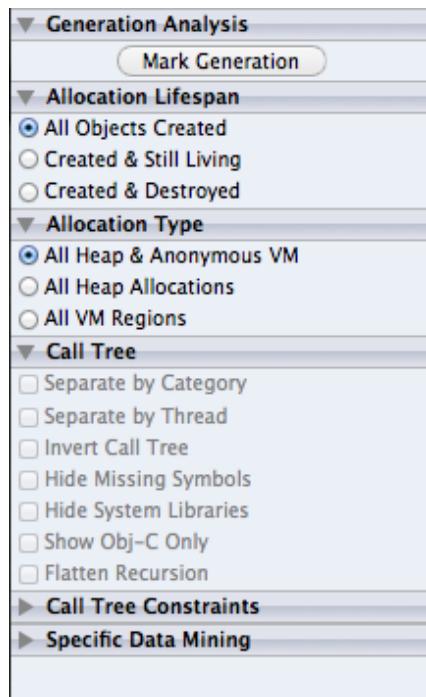
Figure 14.7 Linhas alocadas

| Graph                    | Category | Live Bytes | # Living | # Transient | Overall Bytes | # Overall | Bytes Allocated (Net / Overall) |
|--------------------------|----------|------------|----------|-------------|---------------|-----------|---------------------------------|
| <input type="checkbox"/> | BNRLine  | 128 Bytes  | 4        | 0           | 128 Bytes     | 4         |                                 |

A coluna **# Living** mostra como muitos objetos de linha estão alocados no momento. **Live Bytes** mostra quanta memória essas instâncias ativas estão consumindo. A coluna **# Overall** mostra quantas linhas foram criadas no decorrer do aplicativo – mesmo que já tenham sido liberadas.

Como seria de se esperar, o número de linhas ativas e o número total de linhas são iguais nesse momento. Agora, toque duas vezes em TouchTracker na tela e apague suas linhas. No Instruments, note que as instâncias de **BNRLine** desaparecem da tabela. O instrumento Allocations está definido para mostrar apenas objetos que são criados e ainda estão ativos. Para alterar isso, selecione **All Objects Created** na seção **Allocation Lifespan** do painel esquerdo (Figure 14.8).

Figure 14.8 Opções do Allocations



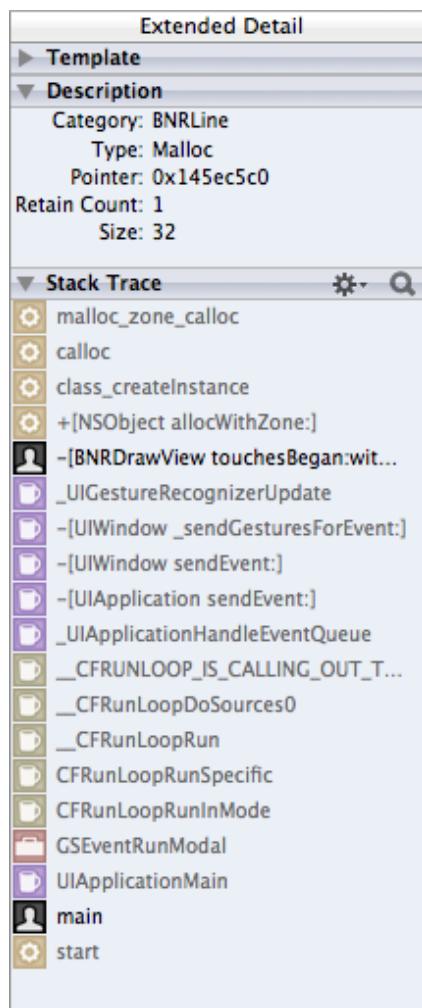
Vejamos o que mais o instrumento Allocations pode dizer a você sobre suas linhas. Primeiramente, desenhe algumas outras linhas no TouchTracker. Em seguida, na tabela, selecione a linha que diz **BNRLine**. Uma seta aparecerá na coluna Category; clique nessa seta para ver mais detalhes sobre essas alocações (Figure 14.9).

Figure 14.9 Resumo da **BNRLine**

| # | Address    | Category | Timestamp     | Live | Size     | Responsible Library | Responsible Caller         |
|---|------------|----------|---------------|------|----------|---------------------|----------------------------|
| 0 | 0x145ec5c0 | BNRLine  | 02:57.539.569 | •    | 32 Bytes | TouchTracker        | -[BNRDrawView touchesBe... |
| 1 | 0x145cf0c0 | BNRLine  | 02:58.552.178 | •    | 32 Bytes | TouchTracker        | -[BNRDrawView touchesBe... |
| 2 | 0x1468a070 | BNRLine  | 02:59.152.489 | •    | 32 Bytes | TouchTracker        | -[BNRDrawView touchesBe... |
| 3 | 0x145cf0a0 | BNRLine  | 03:00.954.103 | •    | 32 Bytes | TouchTracker        | -[BNRDrawView touchesBe... |

Cada linha nessa tabela mostra uma única instância de **BNRLine** que foi criada. Selecione uma das linhas e verifique o rastreamento de pilha que aparece na área Extended Detail, no lado direito da janela do Instruments (Figure 14.10). Esse rastreamento de pilha mostra onde essa instância de **BNRLine** foi criada. Itens em cinza no rastreamento são chamadas feitas à biblioteca do sistema. Itens com texto em preto são o seu código. Encontre o item mais ao topo do seu código (`-[BNRDrawView touchesBegan:withEvent:]`) e clique duas vezes nele.

Figure 14.10 Rastreamento de pilha



O código-fonte para essa implementação substituirá a tabela de instâncias de **BNRLLine** (Figure 14.11). As porcentagens que você vê são a quantidade de memória que essas chamadas de método alocam, em comparação com as outras chamadas no **touchesBegan:withEvent:**. Por exemplo, a instância de **BNRLLine** representa cerca de 0,2% da memória alocada por **touchesBegan:withEvent:**, ao mesmo tempo que **NSLog** aloca uma quantidade considerável de memória em relação à criação da linha, e os dois objetos de **NSNumber** (aquele que você criou e a cópia feita usando-a como chave em **self.linesInProgress**).

Figure 14.11 Código-fonte no aplicativo Instruments

```

Statistics > Allocation Summary > BNRLLine > M -[BNRDrawView touchesBegan:withEvent:]
BNRDrawView.m

159 - (void)touchesBegan:(NSSet *)touches
160         withEvent:(UIEvent *)event
161 {
162     // Let's put in a log statement to see the order of
163     // events
164     NSLog(@"%@", NSStringFromSelector(_cmd));
165     for(UITouch *t in touches) {
166         CGPoint location = [t locationInView:self];
167         BNRLLine *line = [[BNRLLine alloc] init];
168         line.begin = location;
169         line.end = location;
170         NSNumber *key = [NSNumber valueWithNonretainedObject:t];
171         self.linesInProgress[key] = line;
172     }
173     [self setNeedsDisplay];
174 }
175

```

The screenshot shows the 'Allocation Summary' view in Instruments for the BNRLLine category. It displays the source code for the **-[BNRDrawView touchesBegan:withEvent:]** method. The code includes several allocations: an NSLog statement (99.0% of memory), the creation of a BNRLLine object (0.2%), and the creation of an NSNumber object (0.1%). The code also shows the assignment of these objects to a dictionary (**self.linesInProgress**). The memory usage for the entire block of code is highlighted with a red bar.

Observe que acima da área de resumo encontra-se uma barra de trilha de navegação (Figure 14.12). Você pode clicar em um item dessa barra para voltar a um conjunto de informações anterior.

Figure 14.12 Navegação na área de resumo



Clique no item **BNRLLine**, na barra de trilha de navegação, para voltar à lista de todas as instâncias de **BNRLLine**. Clique em uma única instância e, em seguida, clique no ícone de seta nessa linha. Isso mostrará o histórico desse objeto. Há dois eventos: quando a **BNRLLine** foi criada e quando foi destruída. Você pode selecionar uma linha de evento para ver o rastreamento de pilha que resultou no evento, na área de detalhes ampliada.

## Análise de geração

O último item que analisaremos no instrumento Allocations é Generation Analysis (também conhecido como Análise de Heapshot). Primeiro, limpe a caixa de pesquisa para não filtrar mais os resultados. Em seguida, encontre a categoria Generation Analysis no lado esquerdo da janela Instruments e clique em **Mark Generation**. Uma categoria chamada Generation A aparecerá na tabela. Você pode clicar no botão de divulgação ao lado dessa categoria para ver todas as alocações que ocorreram antes de você marcar a geração. Agora, trace uma linha no TouchTracker e clique em **Mark Generation** novamente. Outra categoria aparecerá, chamada Generation B. Clique no botão de divulgação ao lado de Generation B (Figure 14.13).

Figure 14.13 Análise de geração

| Generation                     | Timestamp     | Growth    | # Persistent |
|--------------------------------|---------------|-----------|--------------|
| Generation A                   | 00:06.670.257 | 4.08 MB   | 8,736        |
| Generation B                   | 00:15.766.416 | 1.48 KB   | 30           |
| >< non-object >                |               | 776 Bytes | 13           |
| IOHIDEEvent                    |               | 320 Bytes | 2            |
| CFArray (store-deque)          |               | 96 Bytes  | 3            |
| CFArray (mutable-variable)     |               | 96 Bytes  | 3            |
| _UIGestureRecognizerFailureMap |               | 64 Bytes  | 2            |
| CFDictionary (mutable)         |               | 48 Bytes  | 1            |
| CFBasicHash (key-store)        |               | 32 Bytes  | 2            |
| CFBasicHash (value-store)      |               | 32 Bytes  | 2            |
| BNRLLine                       |               | 32 Bytes  | 1            |
| CGRegion                       |               | 16 Bytes  | 1            |

Cada alocação que ocorreu após a primeira geração está nessa categoria. É possível ver as instâncias de **BNRLLine** que você acabou de criar, bem como alguns objetos que foram usados para lidar com outras partes de código nesse período. Você pode marcar quantas gerações quiser. Elas são muito úteis quando se quer ver quais objetos são alocados para um evento específico. Toque duas vezes em TouchTracker na tela para limpar as linhas e observe que os objetos dessa geração desaparecem.

A análise de geração é muito útil na identificação de tendências de utilização de memória criando um teste de circuito fechado e repetindo-o enquanto é feita a marcação da geração após cada iteração. Por exemplo, você pode traçar quatro linhas e tocar duas vezes nelas para dispensá-las, e então marcar uma geração. Desenhe mais quatro linhas, dispense-as e marque outra geração.

Em um mundo perfeito, você veria alocações de valor nulo ainda ativas entre as duas gerações. Na realidade, haverá diversas pequenas alocações de frameworks da Apple. Você precisa se preocupar apenas com seus próprios objetos. Se, por exemplo, você perceber que suas linhas não estão se desalocando entre as gerações, isso representa um problema (vazamento de memória) e você precisa corrigir.

Para voltar à lista completa de objetos onde você começou, selecione o botão pop-up na barra de trilha de navegação que diz **Generations** e altere-o para **Statistics**.

## Instrumento Time Profiler

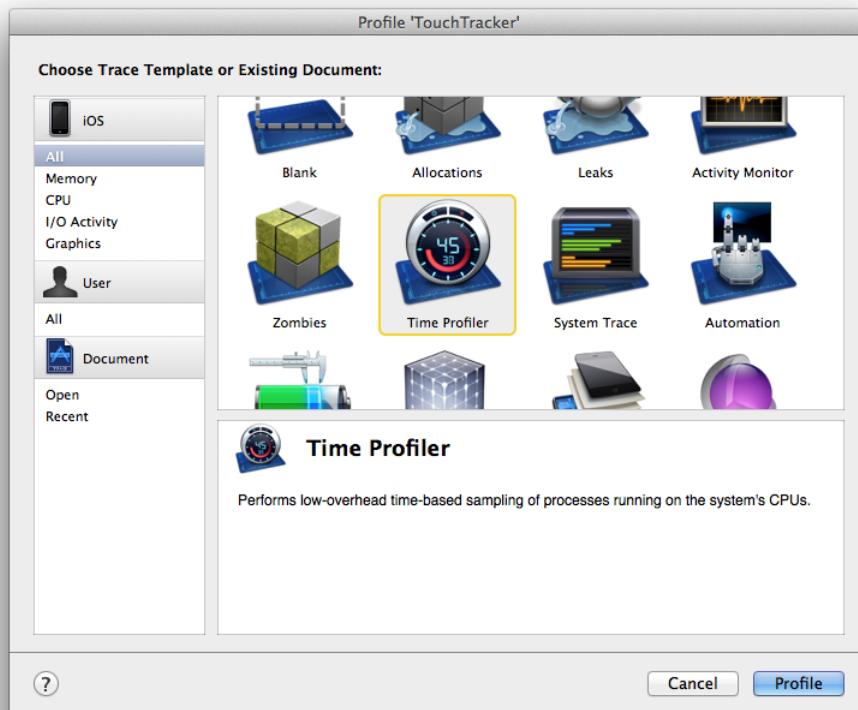
O instrumento Time Profiler oferece estatísticas exaustivas da utilização da CPU por seu aplicativo. Neste momento, o TouchTracker não abusa da CPU o bastante para oferecer resultados significativos.

No `BNRDrawView.m`, torne as coisas mais interessantes adicionando o seguinte código que desperdiça ciclos de CPU no final de seu método `drawRect:`:

```
float f = 0.0;
for (int i = 0; i < 1000000; i++) {
    f = f + sin(sin(sin(time(NULL) + i)));
}
NSLog(@"%@", f);
```

Compile e crie perfis do aplicativo. Quando o Instruments perguntar que instrumento usar, escolha o Time Profiler (Figure 14.14). Quando o Instruments inicializar o aplicativo e sua janela for exibida, verifique se as três áreas estão visíveis, clicando nos botões do controle View até ficarem azuis.

Figure 14.14 Instrumento Time Profiler



Toque e mantenha o dedo na tela do TouchTracker. Mova o dedo ao redor, mas mantenha-o na tela. Isso envia o **touchesMoved:withEvent:** repetidamente para a **BNRDrawView**. Cada mensagem **touchesMoved:withEvent:** faz com que o **drawRect:** seja enviado, o que, por sua vez, faz com que o código **sin** seja executado repetidamente.

Parece que não estão acontecendo muitas coisas nesse instrumento, mas isso se dá porque você está olhando para ele do ângulo errado: você vê apenas quanto tempo é gasto em cada um dos threads que esse aplicativo está empregando. Clique no botão de pausa no canto superior esquerdo do Instruments e então, no painel esquerdo, marque a caixa denominada **Invert Call Tree**. Cada linha na tabela é agora uma chamada de função ou método. Na coluna da esquerda, é exibida a quantidade de tempo gasto naquela função (expresso em milésimos de segundos e como uma porcentagem do tempo total de execução) (Figure 14.15). Isso lhe dá uma ideia de onde seu aplicativo está gastando seu tempo de execução.

Figure 14.15 Resultados do Time Profiler

| Call Tree |              |       |                                                                           |
|-----------|--------------|-------|---------------------------------------------------------------------------|
|           | Running Time | Self  | Symbol Name                                                               |
| 138.0ms   | 22.8%        | 138.0 | ►mach_absolute_time libsystem_kernel.dylib                                |
| 89.0ms    | 14.7%        | 89.0  | ►cos libsystem_m.dylib                                                    |
| 75.0ms    | 12.4%        | 75.0  | ►sin libsystem_m.dylib                                                    |
| 55.0ms    | 9.1%         | 55.0  | ►-[BNRDrawView drawRect:] TouchTracker                                    |
| 25.0ms    | 4.1%         | 25.0  | ►fegetenv libsystem_m.dylib                                               |
| 12.0ms    | 1.9%         | 12.0  | ►search_method_list(method_list_t const*, objc_selector*) libobjc.A.dylib |
| 11.0ms    | 1.8%         | 11.0  | ►_commpage_gettimeofday libsystem_kernel.dylib                            |
| 10.0ms    | 1.6%         | 10.0  | ►fesetenv libsystem_m.dylib                                               |
| 6.0ms     | 0.9%         | 6.0   | ►OSAtomicCompareAndSwap64Barrier libsystem_platform.dylib                 |
| 6.0ms     | 0.9%         | 6.0   | ►_platform_memset_pattern4\$VARIANT\$Swift libsystem_platform.dylib       |
| 5.0ms     | 0.8%         | 5.0   | ►time libsystem_c.dylib                                                   |
| 5.0ms     | 0.8%         | 5.0   | ►_platform_memset\$VARIANT\$Swift libsystem_platform.dylib                |
| 5.0ms     | 0.8%         | 5.0   | ►objc_msgSend libobjc.A.dylib                                             |
| 4.0ms     | 0.6%         | 4.0   | ►_read_images libobjc.A.dylib                                             |
| 4.0ms     | 0.6%         | 4.0   | ►mach_msg_trap libsystem_kernel.dylib                                     |
| 4.0ms     | 0.6%         | 4.0   | ►strcmp dyld                                                              |

Não há uma regra que diga: “Se X% do tempo é gasto em tal função, seu aplicativo está com problema”. Em vez disso, use o Time Profiler se perceber que o aplicativo está lento quando testá-lo como usuário. Por exemplo, você deve notar que desenhar no TouchTracker ficou mais lento desde que você adicionou o código de desperdício `sin`.

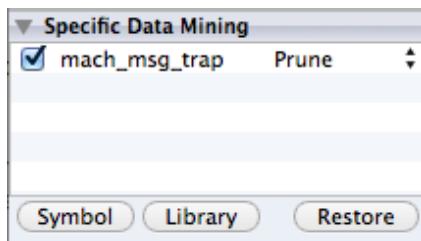
Você sabe que ao traçar uma linha, duas coisas estão acontecendo: `touchesMoved:withEvent:` e `drawRect:` estão sendo enviados para a visão `BNRDrawView`. No Time Profiler, você pode verificar quanto tempo é gasto nesses dois métodos, em relação ao resto do aplicativo. Se uma quantidade excessiva de tempo é gasta com um desses métodos, você sabe que é aí que o problema se encontra.

(Não se esqueça que algumas tarefas simplesmente demoram. Redesenhar a tela inteira sempre que o dedo do usuário se move, como é feito no TouchTracker, é uma operação dispendiosa. Se estivesse prejudicando a experiência do usuário, você poderia encontrar uma maneira de diminuir o número de vezes que a tela é redesenhada. Por exemplo, você poderia redesenhar apenas a cada décimo de segundo, não importando quantos eventos de toque sejam enviados.)

O Time Profiler mostra praticamente todas as chamadas de função e de método do aplicativo. Caso queira focar em determinadas partes do código do aplicativo, você poderá filtrar seus resultados. Por exemplo, às vezes a função `mach_msg_trap` estará bem no topo da lista de exemplos. Essa função é onde o segmento principal reside enquanto aguarda a entrada de dados. Gastar tempo nessa função não significa algo ruim, portanto, você pode decidir ignorar esse tempo ao analisar os resultados do Time Profiler.

Utilize a caixa de pesquisa, no canto superior direito da janela do Instruments, para encontrar a função `mach_msg_trap()`. Em seguida, selecione-a na tabela. No lado esquerdo da tela, clique no botão **Symbol** em Specific Data Mining. A função `mach_msg_trap` aparece na tabela, em Specific Data Mining, e o botão pop-up ao lado exibe Charge. Clique em Charge e mude-o para Prune. Em seguida, limpe o texto da caixa de pesquisa. Agora, a lista é ajustada para que qualquer tempo gasto na função `mach_msg_trap()` seja ignorado. Você pode clicar em Restore enquanto a `mach_msg_trap()` está selecionada na tabela Specific Data Mining para adicioná-la novamente à contagem do tempo total.

Figure 14.16 Supressão de símbolos



Outras opções para reduzir a lista de símbolos no Time Profiler incluem exibir apenas chamadas do Objective-C, ocultar bibliotecas do sistema e encarregar os autores das chamadas. As duas primeiras são óbvias, mas vamos analisar a opção de encarregar os autores das chamadas. Selecione a linha que contém `mach_absolute_time()` (ou algum método que comece com esse nome). Em seguida, clique no botão Symbol. Essa função desaparece da tabela principal e reaparece na tabela Specific Data Mining. Note que ela é listada como Charge. Isso significa que o tempo gasto nessa função será atribuído à função ou método que a chamou.

De volta à tabela principal, observe que `mach_absolute_time()` foi substituída pela função que a chama, `gettimeofday()`. Se você seguir os mesmos passos para encarregar `gettimeofday()`, ela será substituída pela função que a chama, `time()`. Se encarregar `time()`, ela será substituída pelo método que a chama, `drawRect:`. O método `drawRect:` passará para perto do topo da lista. Agora ele está encarregado das funções `time()`, `gettimeofday()` e `mach_absolute_time()`.

Algumas chamadas comuns de função sempre consomem muito tempo da CPU. Na maioria das vezes, elas são inofensivas e inevitáveis. Por exemplo, `objc_msgSend()` é a função central de envio de qualquer mensagem do Objective-C. Ela ocasionalmente chega ao topo da lista quando você está enviando várias mensagens a objetos. Normalmente, não há nada com que se preocupar. No entanto, se você está gastando mais tempo enviando mensagens do que trabalhando nos métodos disparados e o seu aplicativo não está com bom desempenho, você tem um problema que precisa ser resolvido.

Como exemplo, um desenvolvedor em Objective-C com excesso de zelo pode ficar tentado a criar classes para coisas como vetores, pontos e retângulos. Essas classes provavelmente teriam métodos para adicionar,

subtrair ou multiplicar instâncias, bem como métodos acessores para fazer o get e o set de variáveis de instância. Quando essas classes são usadas para desenhar, o código tem de enviar muitas mensagens para fazer algo simples, como criar dois vetores juntá-los. As mensagens adicionam sobrecarga excessiva, considerando-se a simplicidade da operação. Portanto, a melhor alternativa é criar tipos de dados como esses na forma de estruturas e acessar sua memória diretamente. (É por isso que `CGRect` e `CGPoint` são estruturas e não classes no Objective-C.)

Não se esqueça de remover o código que desperdiça ciclos de CPU do `drawRect:`!

## Instrumento Leaks

Outro instrumento útil é o Leaks. Embora esse instrumento seja menos útil agora que a ARC cuida do gerenciamento de memória, ainda há uma possibilidade de vazamento de memória com ciclo de referência forte. O Leaks pode ajudar você a encontrar ciclos de referências fortes.

Primeiro, você precisa introduzir um ciclo de referência forte em seu aplicativo. Imagine que cada `BNRLine` precisa saber a qual array de linhas ela pertence. Adicione uma nova propriedade a `BNRLine.h`:

```
@property (nonatomic, strong) NSMutableArray *containingArray;
```

No `BNRDrawView.m`, defina a propriedade `containingArray` de toda linha concluída em `touchesEnded:withEvent:`.

```
- (void)touchesEnded:(NSSet *)touches
              withEvent:(UIEvent *)event
{
    // Remove ending touches from dictionary
    for (UITouch *t in touches) {
        NSValue *key = [NSValue valueWithNonretainedObject:t];
        BNRLine *line = self.linesInProgress[key];

        [self.finishedLines addObject:line];
        [self.linesInProgress removeObjectForKey:key];

        line.containingArray = self.finishedLines;
    }

    // Redraw
    [self setNeedsDisplay];
}
```

Finalmente, no `doubleTap:` do `BNRDrawView.m`, transforme em comentário o código que remove todos os objetos da `self.finishedLines` e crie uma nova instância da `NSMutableArray`.

```
- (void)doubleTap:(UIGestureRecognizer *)gr
{
    NSLog(@"Recognized Double Tap");

    [self.linesInProgress removeAllObjects];
    // [self.finishedLines removeAllObjects];

    self.finishedLines = [[NSMutableArray alloc] init];

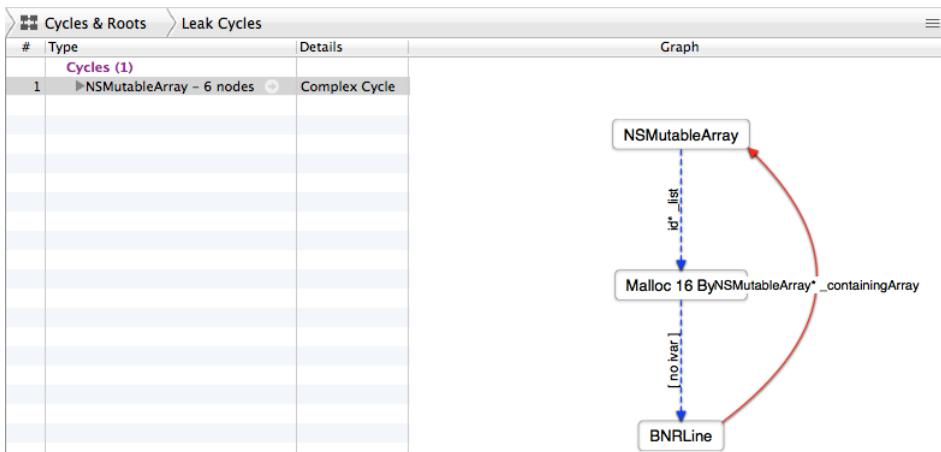
    [self setNeedsDisplay];
}
```

Compile e crie perfis do aplicativo. Escolha o Leaks como instrumento a ser usado.

Trace algumas linhas e, em seguida, toque duas vezes na tela para limpá-la. Selecione o instrumento Leaks na tabela superior da esquerda e espere alguns segundos. Três itens serão exibidos na tabela de resumo: uma `NSMutableArray`, algumas instâncias de `BNRLine` e um bloco `Malloc 16 Bytes`. Essa memória sofreu vazamento.

Selecione o botão pop-up Leaks na barra de trilha de navegação e altere-o para Cycles & Roots (Figure 14.17). Essa visão oferece uma bela representação gráfica do ciclo de referência forte: uma `NSMutableArray` (nossa array `self.finishedLines`) tem uma referência para uma lista de objetos `BNRLine` e cada `BNRLine` tem referência de volta para sua `containingArray`.

Figure 14.17 Ciclos e raízes



É claro que você pode corrigir esse problema transformando a propriedade `containingArray` em uma referência fraca. Ou simplesmente removendo a propriedade e desfazendo as alterações de `touchesEnded:withEvent` e `doubleTap:`.

Isso deve lhe dar um bom começo com o aplicativo Instruments. Quanto mais você interagir com ele, mais apto a usá-lo você se tornará. Uma última palavra antes de você investir parte significativa de seu tempo de desenvolvimento usando o Instruments: caso não haja problemas de desempenho, não se preocupe em analisar cada linha do Instruments. Ele é uma ferramenta para diagnosticar problemas existentes, não para encontrar problemas novos. Primeiro, escreva um código limpo que funcione; depois, se houver um problema, você poderá localizá-lo e corrigi-lo com a ajuda do Instruments.

## Analisador estático

O Instruments pode ser útil quando você tenta detectar problemas em um aplicativo em execução. Você também pode solicitar ao Xcode a análise de seu código sem executá-lo. O analisador estático é uma ferramenta que faz suposições informadas sobre o que aconteceria se seu código fosse executado, informando você quanto a potenciais problemas.

Quando o analisador estático verifica o código, ele examina cada função e método individualmente, fazendo iteração em cada *caminho de código* possível. Um método pode ter algumas instruções de controle (`if`, `for`, `switch` etc.). As condições dessas instruções vão ditar qual código é realmente executado. Um caminho de código é um dos caminhos possíveis que o código tomará, conforme essas instruções de controle. Por exemplo, um método que tem uma única instrução `if` tem dois caminhos de código: um se a condição falhar e outro se a condição tiver êxito.

Neste momento, o TouchTracker não tem nenhum código que viola o analisador estático. Adicione algo: no `BNRDrawView.m`, implemente o seguinte método:

```
- (int)numberOfLines
{
    int count;

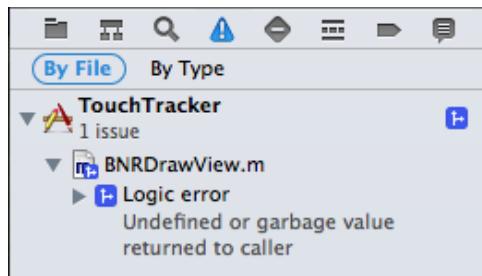
    // Check that they are non-nil before we add their counts...
    if (self.linesInProgress && self.finishedLines)
        count = [self.linesInProgress count] + [self.finishedLines count];

    return count;
}
```

Para executar o analisador estático, mantenha pressionado o botão Run (como fez quando criou o perfil do aplicativo). Desta vez, escolha Analyze. Alternativamente, você pode usar o atalho do teclado Command-Shift-B.

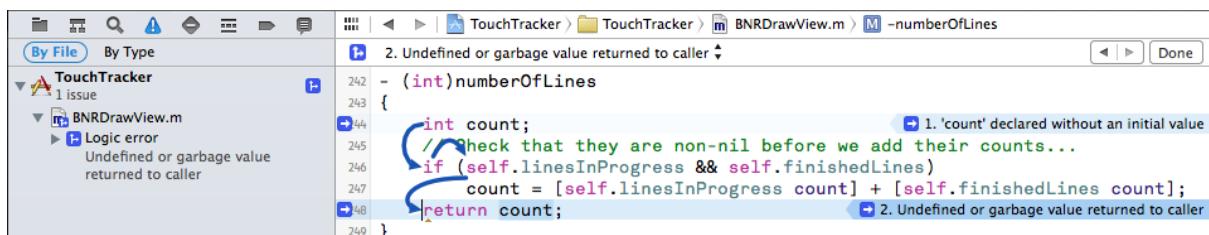
Os resultados da análise aparecem no navegador de problemas (Figure 14.18). Você verá um Logic error no código, no ponto de retorno de `numberOfLines`. O analisador acredita que há um caminho de código que resultará em um valor indefinido ou inválido sendo devolvido ao autor da chamada. Em inglês, isso significa que é possível que a variável `count` não receba um valor antes de retornar de `numberOfLines`.

Figure 14.18 Resultados do analisador



O analisador consegue mostrar como chegou a essa conclusão. Clique no botão de divulgação ao lado do resultado do analisador para revelar as informações detalhadas abaixo dele. Clique no item abaixo do botão de divulgação. Na área do editor, linhas curvas azuis aparecerão dentro do método `numberOfLines` (Figure 14.19). (Caso não esteja vendo os números de linha na medianiz, você pode ativá-los selecionando Preferences no menu do Xcode. Escolha a guia Text Editing e clique na caixa de seleção Show Line Numbers.)

Figure 14.19 Análise expandida



O caminho de código mostrado pelas linhas do analisador é o seguinte:

1. A variável `count` é criada e não é inicializada.
2. A instrução `if` falha, assim, `count` não recebe um valor.
3. A variável `count` é retornada sem valor atribuído.

Você pode corrigir esse problema inicializando a variável `count` com zero.

```
{
    int count;
    int count = 0;

    // Check that they are non-nil before we add their counts...
    if (self.linesInProgress && self.finishedLines)
        count = [self.linesInProgress count] + [self.finishedLines count];

    return count;
}
```

Analise o código novamente e nenhum problema será informado, agora que `count` é sempre inicializada com um valor.

Quando você analisar seu código (o que programadores inteligentes fazem regularmente), verá problemas diferentes daqueles descritos aqui. Muitas vezes, vemos programadores iniciantes esquivarem-se dos problemas do analisador por causa da linguagem técnica. Não faça isso. Dedique um tempo para expandir a análise e entender o que o analisador está tentando lhe dizer. Valerá a pena para o desenvolvimento do seu aplicativo e para o seu desenvolvimento como programador.

## Projetos, destinos e ajustes de compilação

Um projeto do Xcode é um arquivo que contém uma lista de referências para outros arquivos (código-fonte, recursos, frameworks e bibliotecas), bem como inúmeros ajustes que estabelecem regras para itens dentro do projeto. Os projetos terminam no `.xcodeproj` e no `TouchTracker.xcodeproj`.

Os projetos sempre têm, no mínimo, um *destino*. Quando você compila e executa, está compilando e executando o destino, não o projeto. Os destinos usam os arquivos no projeto para criar um determinado *produto*. O produto que o destino compila geralmente é um aplicativo, embora possa ser uma biblioteca compilada ou um pacote de teste de unidade.

Quando você cria um novo projeto e seleciona um template, o Xcode cria um destino automaticamente para você. Ao criar o projeto TouchTracker, você selecionou um template de aplicativo para iOS, portanto, o Xcode criou um aplicativo iOS de destino e o chamou de TouchTracker.

Para ver esse destino, selecione o item TouchTracker na parte superior da lista do navegador de projetos. Na área do editor, bem à direita desse item, localize um botão de alternância (Figure 14.20). Clique nesse botão para exibir a lista de projetos e destinos do TouchTracker.

Figure 14.20 Lista de projetos e destinos do TouchTracker

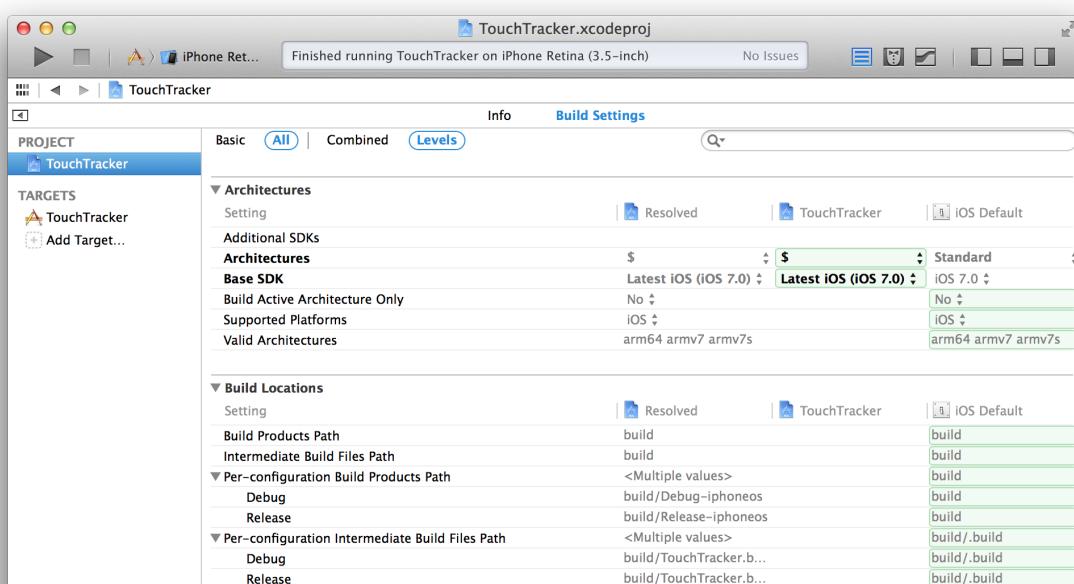
Click to show/hide  
project and targets list



Todo destino inclui *ajustes de compilação* que descrevem como o compilador e o vinculador deveriam compilar seu aplicativo. Todo projeto também tem ajustes de compilação que servem como padrão para os destinos do projeto.

Vamos dar uma olhada primeiro nos ajustes de compilação de projeto do TouchTracker. Na lista de projetos e destinos, selecione o projeto TouchTracker. Em seguida, clique na guia Build Settings, na parte superior da área do editor (Figure 14.21).

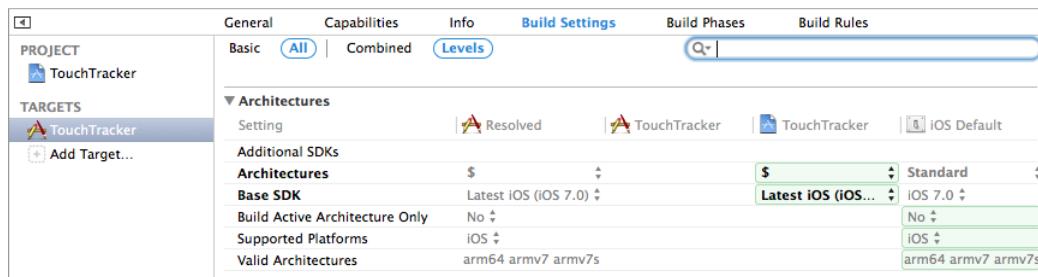
Figure 14.21 Ajustes de compilação do projeto TouchTracker



Esses são os ajustes de compilação em nível de projeto – os valores padrão que os destinos herdam. No canto superior direito encontra-se uma caixa de pesquisa que você pode usar para procurar um ajuste específico. Comece a digitar “Base SDK” na caixa; a lista será ajustada para exibir esse ajuste. (O ajuste Base SDK especifica a versão do SDK do iOS que deveria ser usada na compilação do aplicativo. Ele deve ser sempre definido para a versão mais recente.)

Agora, vamos dar uma olhada nos ajustes de compilação do destino. Na lista de projetos e destinos, selecione o destino TouchTracker e depois a guia Build Settings. Estes são os ajustes de compilação para esse destino específico. Acima da lista de ajustes, encontre e clique na opção Levels (Figure 14.22).

Figure 14.22 Ajustes de compilação do destino TouchTracker



Ao visualizar os ajustes de compilação com essa opção, você pode ver o valor de cada ajuste em três níveis diferentes: sistema operacional (SO), projeto e destino. A coluna mais à direita mostra os ajustes iOS Default; estes servem como o padrão do projeto, podendo ser substituídos. A coluna anterior mostra os ajustes do projeto, e a seguinte mostra os ajustes do destino selecionado no momento. A coluna Resolved mostra qual ajuste será realmente usada; ela é sempre igual ao valor especificado mais à esquerda. Você pode clicar em cada coluna para definir o valor para aquele nível.

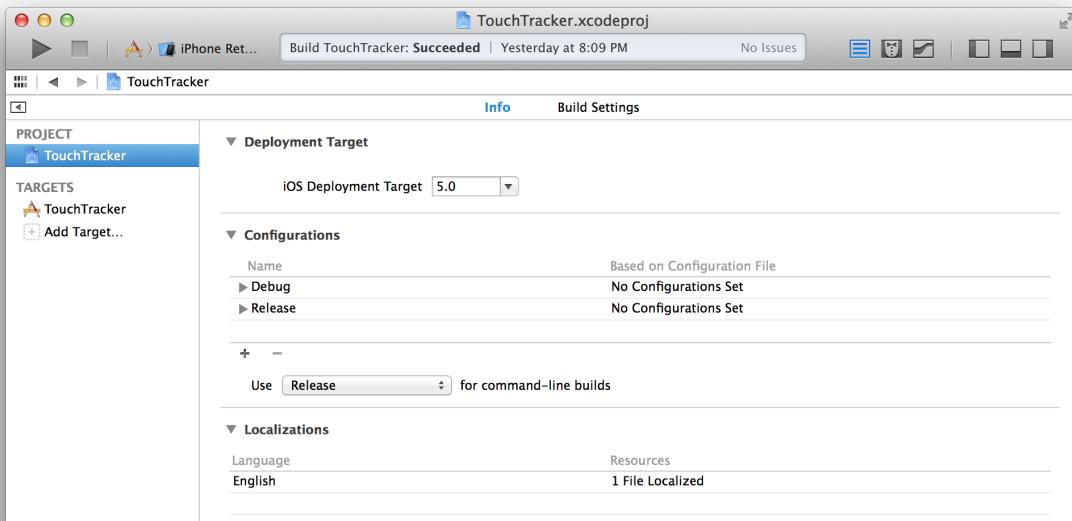
Enquanto você está aqui, pesquise por “analisador estático” na lista de ajustes de compilação. Você pode definir o ajuste Analyze During 'Build' para Yes, para que o Xcode execute automaticamente o analisador estático todas as vezes que você compilar seu aplicativo. A quantidade de tempo de compilação aumentará um pouco, mas essa ideia costuma ser boa.

## Ajustes de compilação

Cada destino e projeto têm vários *ajustes de compilação*. Uma configuração de compilação é um conjunto de ajustes de compilação. Quando você cria um projeto, há duas configurações de compilação: depuração e liberação. Os ajustes de compilação da configuração de depuração facilitam a depuração do seu aplicativo, enquanto os ajustes de liberação ativam otimizações para acelerar a execução.

Para ver as configurações e os ajustes de compilação do TouchTracker, selecione o projeto no navegador de projetos e o projeto do TouchTracker. Depois, selecione a guia Info (Figure 14.23).

Figure 14.23 Lista de configurações de compilação



A seção Configurations mostra as configurações de compilação disponíveis no projeto e nos destinos. Você pode adicionar e remover configurações de compilação com os botões localizados na parte inferior dessa seção.

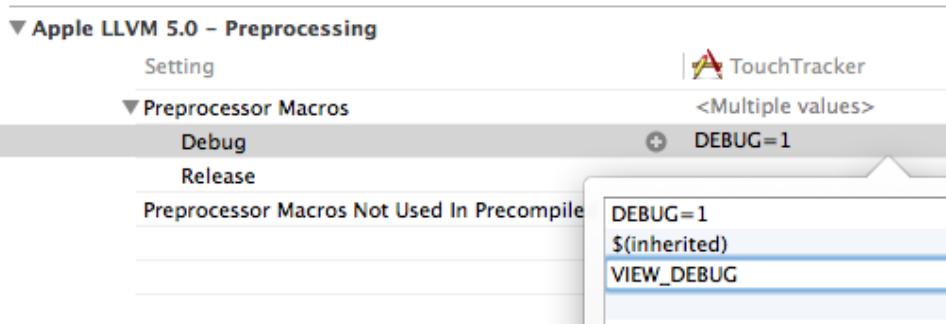
Ao realizar uma ação de esquema, o esquema usará uma dessas configurações ao compilar seus destinos. Você pode especificar a configuração de compilação que o esquema usa no editor de esquemas por meio da opção Build Configuration da guia Info.

## Alteração de um ajuste de compilação

Chega de papo – está na hora de fazer algo útil. Você mudará o valor do ajuste de compilação do destino Preprocessor Macros. Macros de pré-processador permitem compilar códigos de maneira condicional. Elas podem ser definidas ou não no início de uma compilação. Se você encapsular um bloco de código em uma diretiva de pré-processador, ele só será compilado se essa macro tiver sido definida. O ajuste Preprocessor Macros lista as macros de pré-processador que são definidas quando uma determinada configuração de compilação é usada por um esquema para compilar um destino.

Na lista de projetos e destinos, selecione o destino TouchTracker e depois a guia Build Settings. Então, procure o ajuste de compilação Preprocessor Macros. Clique duas vezes na coluna de valor da configuração Debug em Preprocessor Macros. Na tabela que é exibida, adicione um novo item: VIEW\_DEBUG, conforme mostrado na Figure 14.24.

Figure 14.24 Alteração de um ajuste de compilação



Adicionar esse valor a esse ajuste significa: “ao compilar o destino TouchTracker com a configuração de depuração, uma macro de pré-processador VIEW\_DEBUG é definida”.

Vamos adicionar algum código de depuração ao TouchTracker que só será compilado quando o destino for compilado com a configuração de depuração. A **UIView** tem um método particular **recursiveDescription** que mostra toda a hierarquia de visões de um aplicativo. No entanto, não é possível chamar esse método em um aplicativo que você disponibiliza na App Store e, portanto, você só vai permitir que ele seja chamado se a **VIEW\_DEBUG** estiver definida.

No **BNRAppDelegate.m**, adicione o seguinte código ao método **application:didFinishLaunchingWithOptions:**.

```
[self.window makeKeyAndVisible];
#ifndef VIEW_DEBUG
    NSLog(@"%@", [[self window] performSelector:@selector(recursiveDescription)]);
#endif
    return YES;
}
```

Esse código enviará a mensagem **recursiveDescription** para a janela. (Observe o uso do **performSelector:**.) O **recursiveDescription** é um método particular, portanto, você terá de enviá-lo dessa forma.) O **recursiveDescription** mostrará a descrição de uma visão, depois, de todas as suas subvisões e das subvisões de suas subvisões, e assim por diante. Você pode manter esse código para todas as compilações. Uma vez que a macro de pré-processador não será definida para uma compilação de liberação, o código não será compilado quando você compilar para a App Store.

Agora, compile e execute o aplicativo. Verifique o console e você verá a hierarquia de visões do seu aplicativo, começando na janela.



# 15

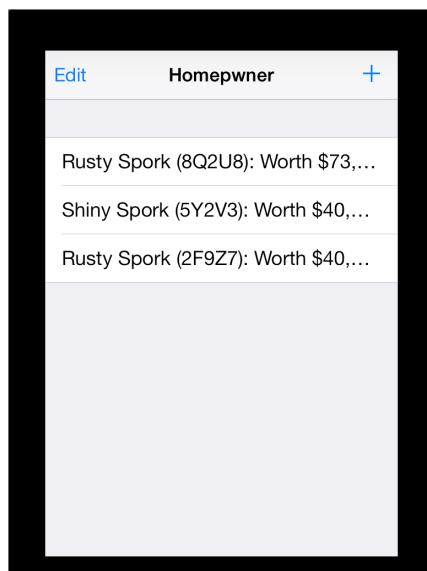
## Introdução ao Auto Layout

Neste capítulo, você retomará ao Homepwner e o universalizará, para que ele seja executado em um iPad ou em um iPhone. Em seguida, você usará o sistema Auto Layout (layout automático) para garantir que a interface de detalhes do Homepwner seja exibida da maneira que você deseja, independentemente do tipo de dispositivo no qual ela está sendo executada.

### Universalização do Homepwner

Atualmente, o Homepwner pode ser executado no simulador de iPad, mas sua aparência não é satisfatória.

Figure 15.1 Aplicativo para iPhone sendo executado em um iPad simulado

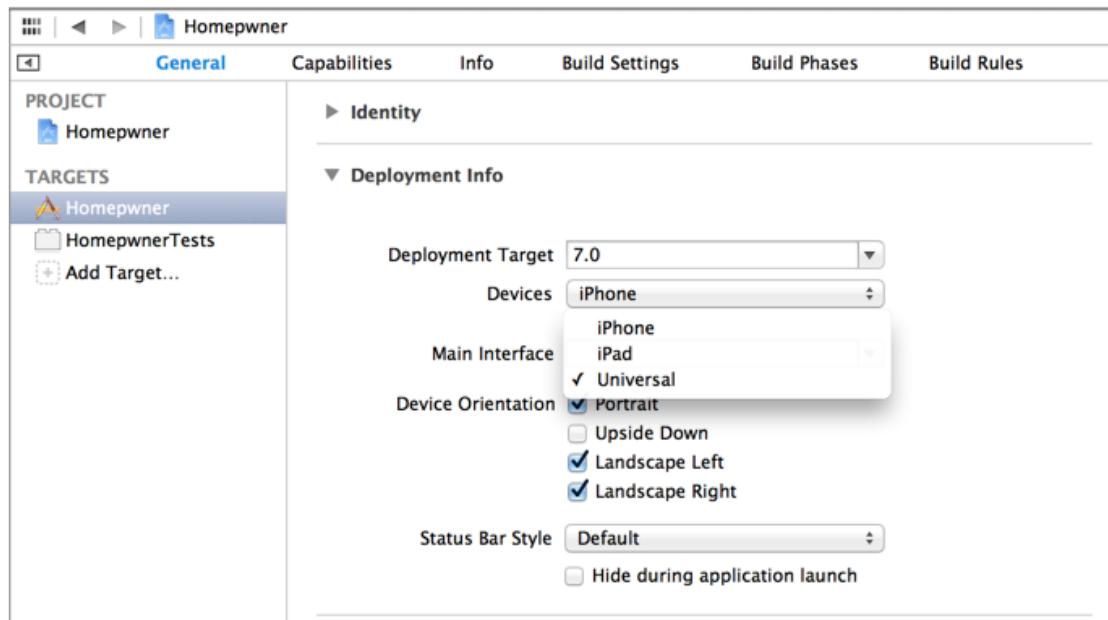


Não é isso que você deseja para seus usuários de iPad. Você deseja que o Homepwner seja executado *nativamente* no iPad, assim parecerá ser um aplicativo para iPad. Um único aplicativo executado nativamente tanto no iPad quanto no iPhone é chamado de *aplicativo universal*.

Abra novamente seu projeto do Homepwner. No navegador de projetos, selecione o projeto Homepwner (o item no topo da lista de arquivos). Em seguida, selecione o destino do Homepwner na lista de projetos e destinos e na guia General. Essa guia conta com uma interface conveniente para edição de algumas propriedades do destino.

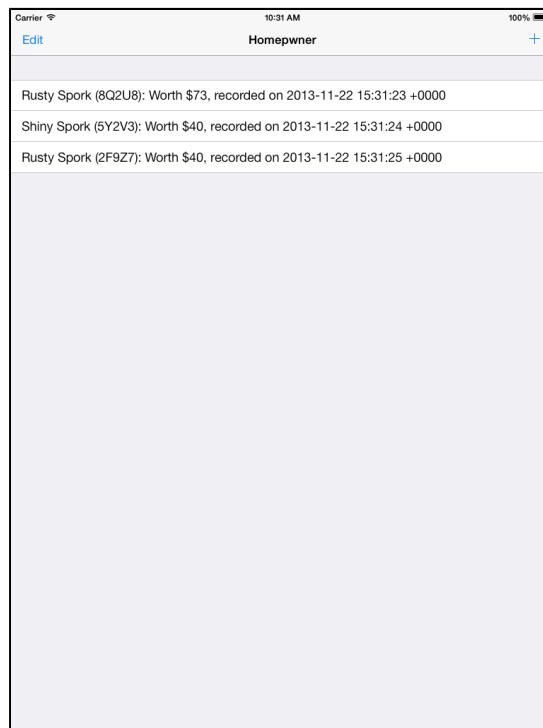
Localize a seção Deployment Info e mude o pop-up Devices de iPhone para Universal (Figure 15.2).

Figure 15.2 Universalização do Homepwner



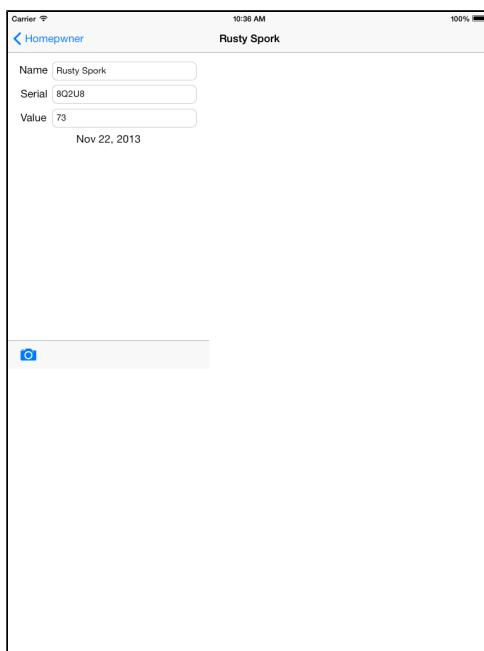
O Homepwner agora é um aplicativo universal. Para ver a diferença, selecione um simulador de iPad no menu pop-up do esquema e compile e execute o aplicativo. O Homepwner parece estar muito melhor. A visão de tabela e suas linhas foram dimensionadas adequadamente na tela do iPad.

Figure 15.3 Aplicativo universal sendo executado no iPad



Agora, adicione um novo item para acessar a interface de detalhes. Estes resultados não são tão bons:

Figure 15.4 Interface de detalhes não é dimensionada automaticamente



A interface da visão de tabela sabe como se redimensionar de acordo com o tamanho da tela do iPad, mas sua interface de detalhes personalizada precisa de algumas instruções. Você fornecerá essas instruções usando o Auto Layout.

## O sistema de layout automático

No Chapter 4, você aprendeu que a variável `frame` da visão especifica seu tamanho e posição em relação à supervisão. Até agora, você definiu os frames de suas visões com coordenadas absolutas de forma programática ou configurando-os no Interface Builder. As coordenadas absolutas, no entanto, tornam seu layout frágil, pois assumem que você sabe qual é o tamanho da tela com antecedência.

Ao utilizar o Auto Layout (layout automático), é possível descrever o layout de suas visões de maneira relativa, a qual permite que os frames sejam determinados no tempo de execução; dessa forma, as definições dos frames podem levar em conta o tamanho da tela do dispositivo no qual o aplicativo está sendo executado.

O tamanho da tela de cada dispositivo está listado na Table 15.1. (Lembre-se de que pontos são usados durante a definição do layout de sua interface, bem como um mapa de pixels físicos na tela do dispositivo. Um dispositivo retina tem o mesmo tamanho de tela em pontos de um dispositivo que não é retina, embora tenha duas vezes mais pixels.)

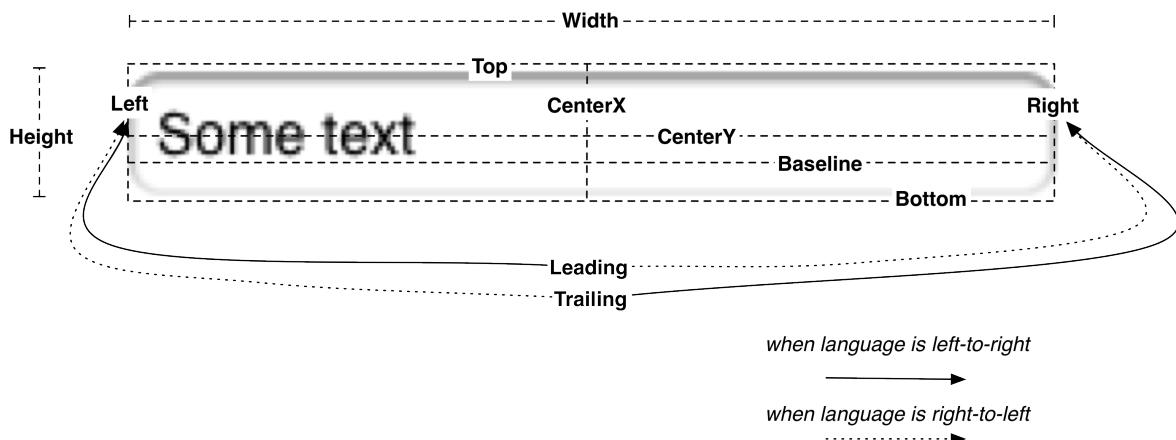
Table 15.1 Tamanhos de tela do dispositivo

| Dispositivo                   | Largura x Altura (pontos) |
|-------------------------------|---------------------------|
| iPhone/iPod (4S e anteriores) | 320 x 480                 |
| iPhone/iPod (5 e posteriores) | 320 x 568                 |
| Todos os iPads                | 768 x 1024                |

## Retângulo de alinhamento e atributos de layout

O sistema de layout automático trabalha ainda com outro retângulo para a visão: o *retângulo de alinhamento*. Esse retângulo é definido por diversos *atributos de layout* (Figure 15.5).

Figure 15.5 Atributos de layout que definem um retângulo de alinhamento de uma visão



## Width/Height

Esses valores determinam o tamanho do retângulo de alinhamento.

## Top/Bottom/Left/Right

Esses valores determinam o espaço entre a borda apresentada do retângulo de alinhamento e o retângulo de alinhamento de outra visão na hierarquia.

## CenterX/CenterY

Esses valores determinam o ponto central do retângulo de alinhamento.

## Baseline

Esse valor é o mesmo do atributo inferior na maioria das visões, mas não em todas. Por exemplo, a linha de base de **UITextField** é definida na parte inferior do texto exibido em vez de ser definida na parte inferior do retângulo de alinhamento. Isso evita que os “descendentes” (letras como “g” e “p” que descendem abaixo da linha de base) fiquem ilegíveis por causa de uma visão bem abaixo do campo de texto.

## Leading/Trailing

Esses valores são usados com visões baseadas em texto, como **UITextField** e **UILabel**. Se a linguagem do dispositivo estiver definida com uma linguagem que lê da esquerda para a direita (p. ex., português), então o atributo líder será o mesmo que o atributo à esquerda, e o atributo final será o mesmo que o atributo à direita. Se a linguagem lê da direita para a esquerda (p. ex., árabe), então o atributo condutor ficará à direita, e o atributo final ficará à esquerda.

Por padrão, toda visão de um arquivo XIB possui um retângulo de alinhamento, e toda hierarquia de visões usa o Auto Layout. Contudo, como você já viu em sua visão de detalhes, o padrão nem sempre funcionará como você deseja. É aí que você precisa intervir.

Não defina um retângulo de alinhamento da visão diretamente. Você não possui informação suficiente (tamanho da tela!) para isso. Em vez disso, forneça um conjunto de *restrições*. Juntas, essas restrições permitem que o sistema determine os atributos de layout e, assim, o retângulo de alinhamento, para cada visão na hierarquia de visões.

## Restrições

Uma *restrição* define um relacionamento específico em uma hierarquia de visão que pode ser usada para determinar um atributo de layout para uma ou mais visões. Por exemplo, você pode adicionar uma restrição como “o espaço vertical entre essas duas visões deve ser sempre 8 pontos” ou “essas visões devem sempre ter a mesma largura”. As restrições também podem ser usadas para dar um tamanho fixo a uma visão, por exemplo, “a altura desta visão deve ser sempre 44 pontos”.

Não é necessário ter uma restrição para cada atributo de layout. Alguns valores podem vir diretamente de uma restrição, outros serão calculados com base nos valores de atributos de layout relacionados. Por exemplo, se as restrições de uma visão definem sua borda esquerda e sua largura, então a borda direita já está determinada (borda esquerda + largura = borda direita, sempre).

Se, após todas as restrições serem consideradas, ainda houver um valor ambíguo ou ausente para um atributo de layout, haverá erros e alertas do Auto Layout e sua interface não ficará como você espera em todos os dispositivos. Depurar esses problemas é importante; você terá exercícios práticos posteriormente neste capítulo.

Como criar restrições? Vamos ver como se usa a visão na hierarquia de visões da **BNRDetailViewController** cuja restrição será simples – a barra de ferramentas.

Primeiro, descreva como você deseja que a visão se pareça, independentemente do tamanho da tela. No caso da barra de ferramentas, você poderia descrevê-la desta forma:

- A barra de ferramentas deverá ficar na parte inferior da tela.
- A barra de ferramentas deve ter a mesma largura da tela.
- A altura da barra de ferramentas deve ser de 44 pontos. (Esse é o padrão da Apple para instâncias de **UIToolbar**.)

Para transformar essa descrição em restrição no Interface Builder, seria de grande ajuda entender como encontrar o *vizinho mais próximo* da visão. O vizinho mais próximo é a visão irmã mais próxima na direção especificada.

Figure 15.6 Vizinho mais próximo



Se uma visão não tiver nenhuma irmã na direção especificada, então o vizinho mais próximo é sua supervisão, também conhecida como seu *recipiente*.

Agora você pode apresentar as restrições para a barra de ferramentas:

1. A borda inferior da barra de ferramentas deve ter 0 ponto de distância de seu vizinho mais próximo (que é seu recipiente – a view da **BNRDetailViewController**).
2. A borda esquerda da barra de ferramentas deve ter 0 ponto de distância de seu vizinho mais próximo.
3. A borda direita da barra de ferramentas deve ter 0 ponto de distância de seu vizinho mais próximo.
4. A altura da barra de ferramentas deve ser de 44 pontos.

Se você considerar a segunda e a terceira restrição, vai poder ver que não há necessidade de restringir explicitamente a largura da barra de ferramentas. Ela será determinada com base nas restrições das bordas

esquerda e direita da barra de ferramentas. Não há necessidade também de restringir a borda superior da barra de ferramentas. As restrições da borda inferior e da altura determinarão o valor do atributo superior.

Agora que você tem um plano para a barra de ferramentas, pode adicionar essas restrições. As restrições podem ser adicionadas por meio do Interface Builder ou em código.

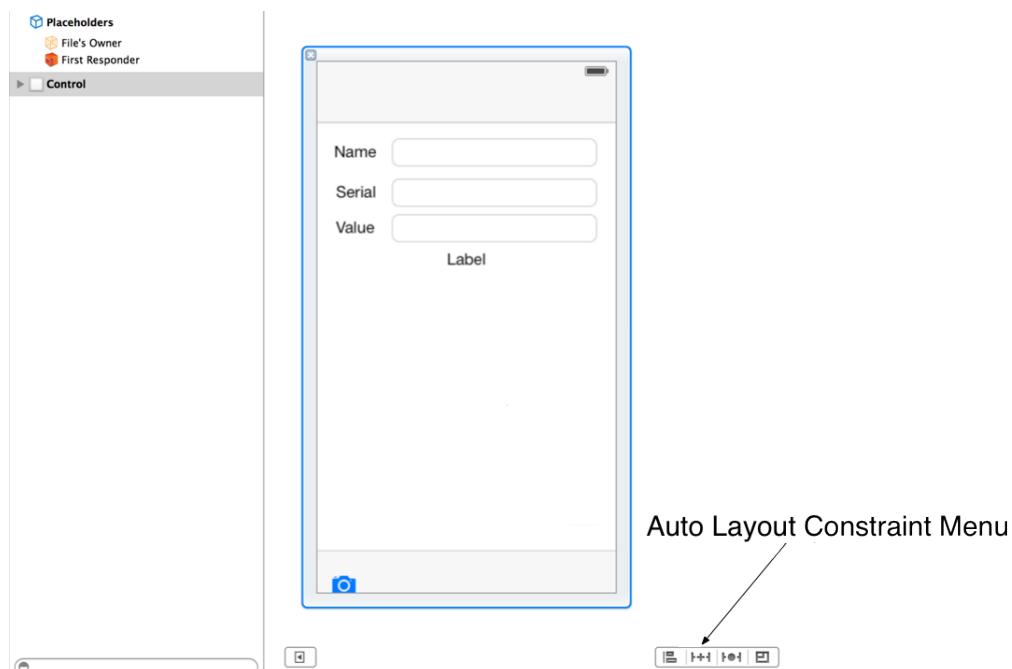
A Apple recomenda que você adicione restrições usando o Interface Builder sempre que possível. Isso é o que você fará neste capítulo. Contudo, se suas visões forem criadas e configuradas de modo programático, você poderá adicionar restrições em código. No Chapter 16, você poderá praticar essa abordagem.

## Adição de restrições no Interface Builder

Abra o `BNRDetailViewController.xib`. Primeiro, selecione a visão de imagem no canvas e exclua-a do arquivo XIB. Você recriará a visão de imagem e adicionará suas restrições de modo programático no Chapter 16.

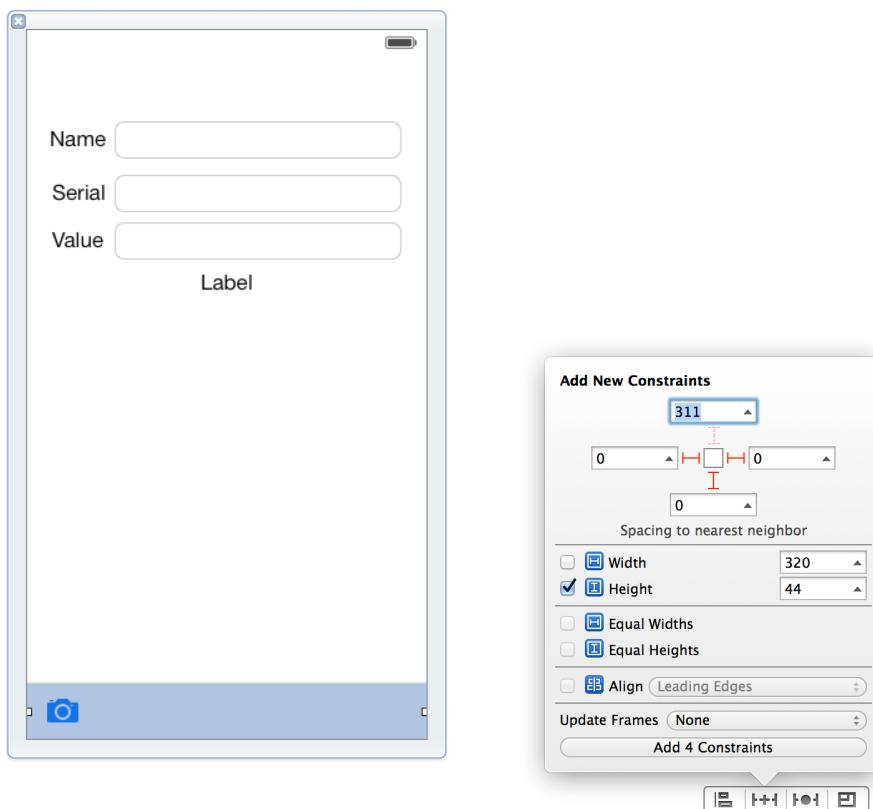
Selecione a barra de ferramentas no canvas. No canto inferior direito do canvas, localize o menu de restrições de layout automático (Figure 15.7).

Figure 15.7 Selecionando uma restrição



Clique no ícone (o segundo à esquerda) para exibir o menu Pin. Esse menu mostra o tamanho atual e a posição da barra de ferramentas (Figure 15.8). É possível adicionar todas as restrições necessárias para a barra de ferramentas nesse menu.

Figure 15.8 Adição de 4 restrições à barra de ferramentas



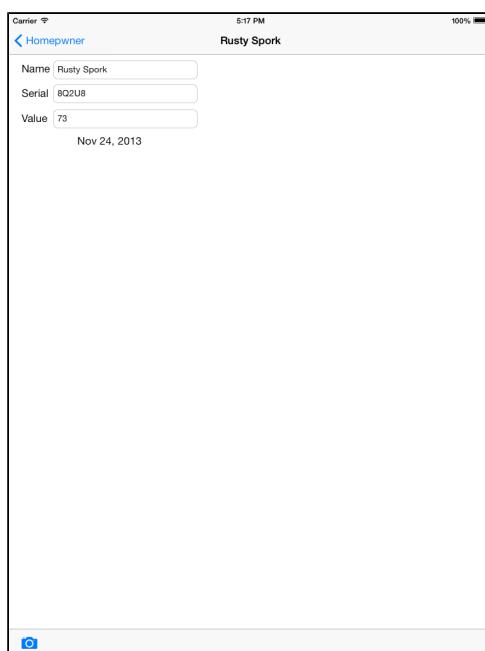
No topo do menu Pin, há quatro valores que descrevem o espaçamento atual da barra de ferramentas de seu vizinho mais próximo no canvas. No caso da barra de ferramentas, você está interessado apenas nos valores da parte inferior, esquerda e direita. Todos são 0, o que significa que essas bordas da barra de ferramentas estão atualmente a 0 ponto de distância do vizinho mais próximo da barra de ferramentas nessas direções. A barra de ferramentas não tem irmãos na parte inferior, esquerda ou direita, portanto, o vizinho mais próximo nas três direções é seu recipiente, a view da **BNRDetailViewController**.

Para transformar esses valores em restrições, clique nos struts laranjas que separam os valores do quadrado no meio. Os struts serão transformados em linhas sólidas.

No meio do menu, localize a Height da barra de ferramentas. Ela atualmente possui 44 pontos, que é o que você deseja. Para restringir a altura da barra de ferramentas com base nesse valor, marque a caixa ao lado de Height. O botão na parte inferior do menu mostra Add 4 Constraints. Clique nesse botão.

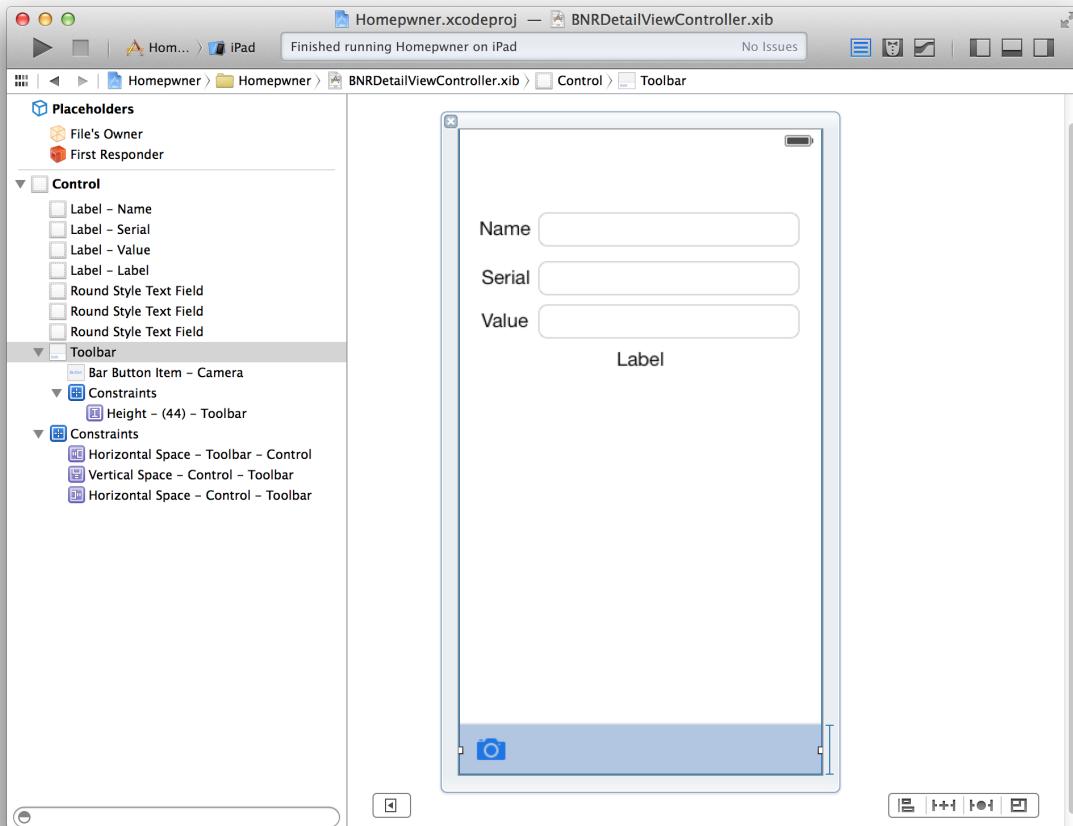
Compile e execute o aplicativo no simulador de iPad. Crie um item e selecione-o para navegar até a interface de detalhes. A barra de ferramentas aparecerá na parte inferior da tela e será da mesma largura que a tela.

Figure 15.9 Barra de ferramentas aparecendo corretamente



É possível ver as restrições que você acabou de adicionar no dock, à esquerda do canvas. Localize a seção **Constraints** e revele o conteúdo. Contudo, você verá apenas três restrições aqui. A quarta restrição, a altura fixada da barra de ferramentas, está em uma seção **Constraints** separada, abaixo de **Toolbar**. Revele o conteúdo dessa seção **Constraints**.

Figure 15.10 Restrições no dock



Por que a divisão? Assim que uma restrição é criada, ela é adicionada a um objeto de visão em particular na hierarquia de visões. A visão que receberá a restrição dependerá das visões que serão afetadas por essa restrição. As três restrições de borda são adicionadas a Control, pois se aplicam tanto à barra de ferramentas quanto a sua supervisão, a view da **BNRDetailViewController**. (Lembre-se de que você alterou a classe desse objeto de visão de **UIView** para **UIControl** no final do Chapter 11 para habilitar o toque nessa visão, dispensando o teclado.) A restrição de altura, por outro lado, é adicionada à barra de ferramentas porque se aplica apenas à barra de ferramentas.

Em um arquivo XIB, a restrição é adicionada automaticamente à visão apropriada. No caso de restrições criadas de forma programática, a criação e a adição são etapas distintas. No próximo capítulo, você verá como determinar qual será a visão à qual a restrição deverá ser adicionada ao criar restrições de forma programática.

Se você selecionar alguma dessas restrições no dock, será exibida uma linha azul no canvas representando a restrição. (Algumas restrições serão mais difíceis de serem visualizadas do que outras.) A seleção da visão mostrará a você todas as restrições que estiverem influenciando tal visão.

É possível excluir restrições selecionando-as no dock ou selecionando sua linha representativa no canvas e pressionando Delete. Tente fazer isso. Exclua a restrição de altura e selecione a barra de ferramentas no canvas e use o menu Pin para adicioná-la de volta.

## Adição de mais restrições

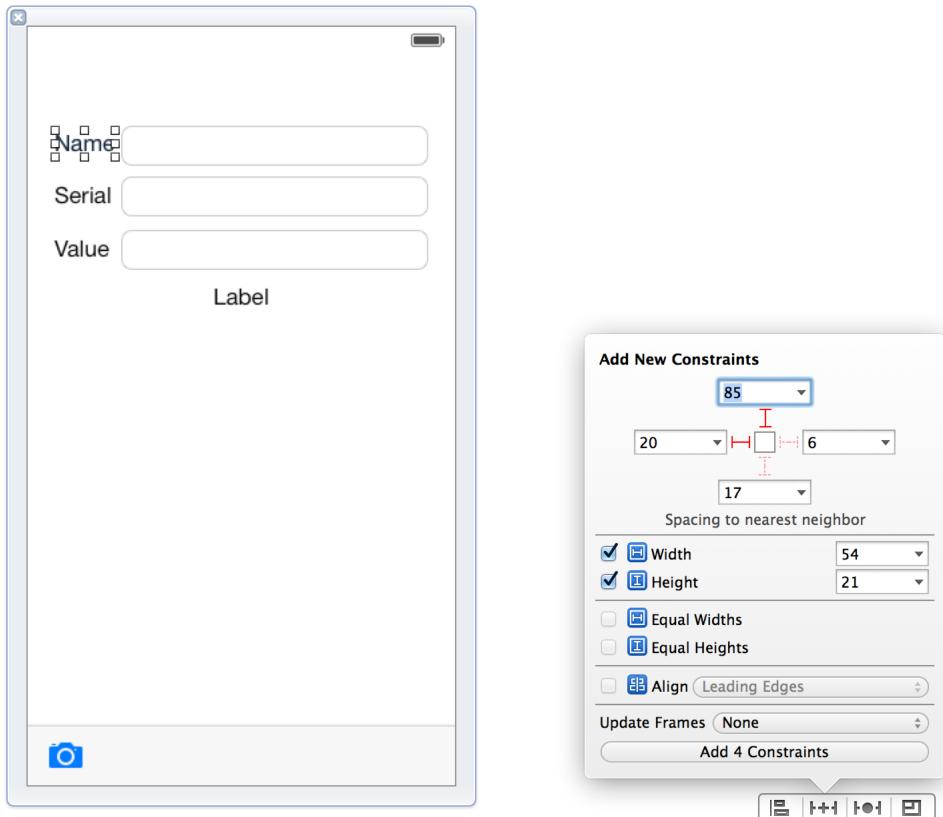
Vejamos agora o rótulo Name. O tamanho e a posição do rótulo estão bons agora quando ele é executado no iPad. Contudo, você ainda precisa restringir essa visão, para que ela apareça adequadamente caso seja apresentada em um idioma ou tamanho de fonte diferente.

Independentemente do idioma, do tamanho da fonte ou do tamanho da tela, você deseja que o tamanho do rótulo Name seja fixado e que sua posição seja próxima ao canto superior esquerdo. Selecione o rótulo Name no canvas e clique no botão de menu Pin no menu de restrições.

Selecione o strut superior e o esquerdo no topo do menu de fixação (Pin). O vizinho mais próximo nessas direções é o recipiente do rótulo Name (também a view da **BNRDetailViewController**). Além disso, marque as caixas Width e Height para corrigir o tamanho atual em pontos do rótulo.

Seu menu Pin deve ficar mais ou menos parecido com a Figure 15.11. Observe que os valores provavelmente não serão correspondentes exatos aos dessa figura, mas tudo bem. Você está criando uma restrição com base na posição da visão em *seu* canvas. Se você alterar seus valores de modo a corresponderem com a Figure 15.11, suas restrições não corresponderão à posição de suas visões no canvas, e você receberá avisos de visão incorretamente posicionada. Você aprenderá sobre esses avisos em breve, mas é melhor evitá-los por enquanto.

Figure 15.11 Exemplos de restrições para o rótulo Name



Clique em Add 4 Constraints na parte inferior do menu.

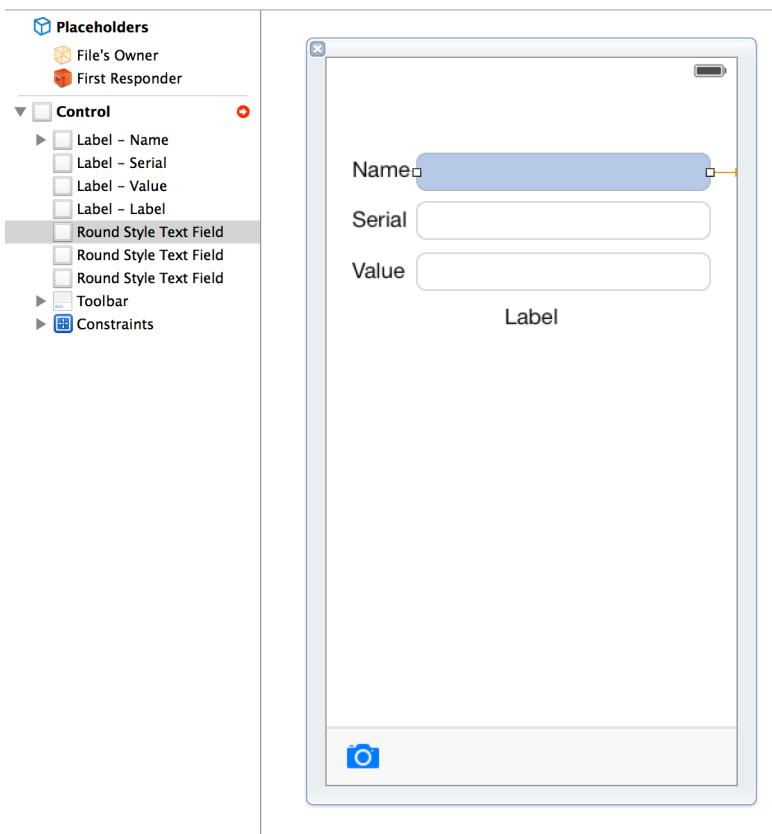
Agora considere o campo de texto à direita do rótulo Name. Independentemente do tamanho da tela, a largura do campo de texto deverá se estender de sua posição à esquerda do rótulo Name e preencher boa parte da tela.

Selecione o campo de texto e abra o menu Pin. No topo do menu, selecione os struts esquerdo e direito e adicione essas duas restrições. A borda anterior (esquerda) do campo de texto será fixada ao rótulo Name e a borda posterior (direita) a 20 pontos do recipiente. Ao fixar a borda posterior (direita) do campo de texto ao seu recipiente, você está se certificando de que o campo de texto sempre se estenderá e preencherá boa parte da tela, independentemente do tamanho dela.

Você agora tem um problema. Observe que as linhas que representam as restrições no campo de texto estão na cor laranja em vez de azul. Essa diferença de cor significa que o campo de texto não tem restrições suficientes para o Auto Layout, para especificar sem ambiguidade seu retângulo de alinhamento.

Para obter mais informações sobre esse problema, localize e clique no ícone vermelho no dock próximo a Control.

Figure 15.12 Restrições insuficientes para o campo de texto

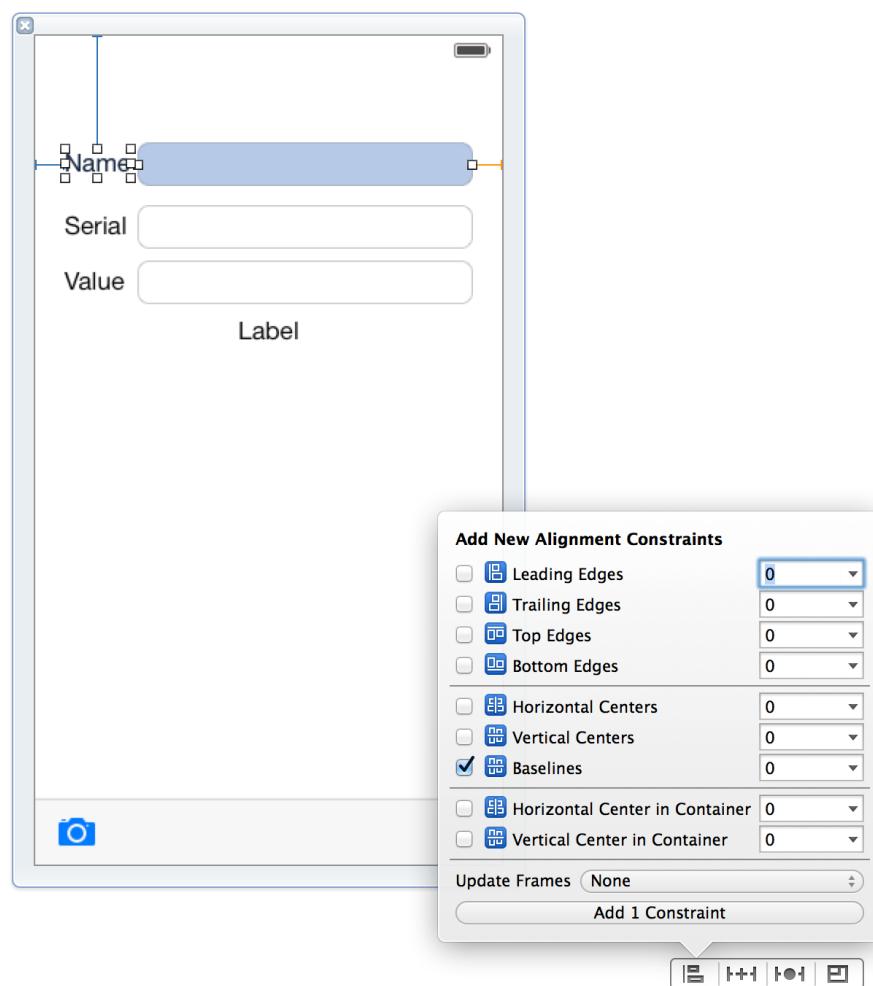


De acordo com o Interface Builder, está faltando uma restrição. O campo de texto precisa que sua posição Y (vertical) seja restringida. Você pode corrigir esse problema abrindo o menu Pin e selecionando o strut superior. O campo de texto será fixado a certa distância de seu recipiente.

Há uma opção de design melhor: alinhe o campo de texto ao rótulo Name, em vez disso. É possível alinhar duas ou mais visões de acordo com qualquer atributo de layout. Aqui, a melhor opção é alinhar as linhas de base. Dessa forma, o texto que o usuário inserir no campo ficará alinhado ao texto do rótulo Name.

No canvas, selecione o campo de texto e mantenha pressionada a tela Shift para selecionar o rótulo Name ao mesmo tempo. No menu de restrições, clique no ícone para revelar o menu Align. Marque a caixa ao lado de Baselines e adicione 1 restrição (Figure 15.13).

Figure 15.13 Alinhamento de linhas de base de rótulo e campo de texto

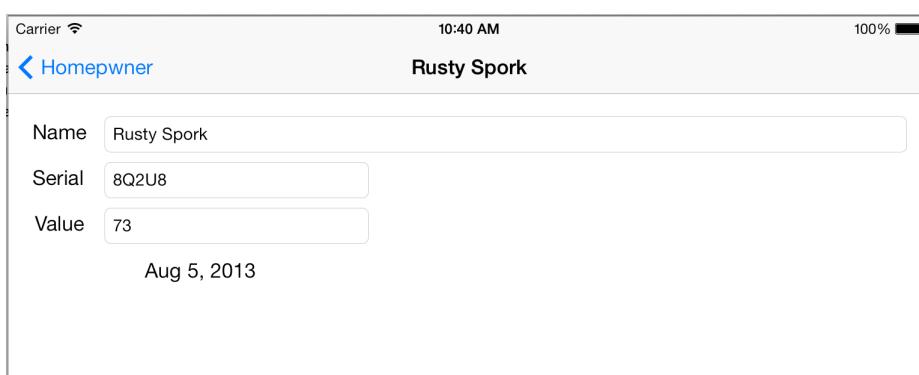


Agora, as linhas que representam as restrições estão azuis novamente, e o ícone vermelho desapareceu; o campo de texto possui restrições suficientes para ser dimensionado e posicionado sem ambiguidade.

Uma restrição ausente (também chamada de layout ambíguo) é apenas um dos problemas que podem surgir quando se adiciona restrições. Dois outros tipos são as restrições conflitantes e as restrições não correspondentes ao tamanho e à posição da visão no canvas. Posteriormente neste capítulo, você saberá o que fazer para depurar esses problemas.

Compile e execute o projeto no iPad. Selecione um item; você deve ver o campo de texto superior se estendendo até preencher o espaço extra deixado pelo tamanho da tela do iPad (Figure 15.14).

Figure 15.14 Estendendo o campo de nome



## Adição de ainda mais restrições

Agora que você conhece as restrições de fixação e alinhamento, você pode adicionar restrições no restante das visões, começando pelo rótulo Serial.

Estas são as restrições que você precisa adicionar para o rótulo Serial:

- a borda superior deve ser fixada à distância atual do rótulo Name
- a borda anterior (esquerda) deve estar alinhada à borda anterior do rótulo Name
- a altura e a largura devem ser fixadas em seus valores atuais

Até aqui, você adicionou restrições usando os menus Pin e Align. Você também pode adicionar restrições pressionando a tela Control e arrastando-as no canvas. Esse arraste é similar à configuração de outlets e ações. Arraste-as de uma visão para outra. Após soltar o botão do mouse, você verá uma lista de restrições que poderá adicionar. A lista é sensível ao contexto e é preenchida com base na direção do arraste e nas visões das quais e para as quais você está fazendo arraste.

Vejamos como funciona o arraste com a tecla Control. Selecione o rótulo Serial no canvas. Em seguida, segure a tecla Control e arraste as restrições desse rótulo para o rótulo Name. Quando você soltar o botão do mouse, um menu será exibido (Figure 15.15).

Figure 15.15 Adição de restrição pressionando a tecla Control e arrastando entre visões



Selecione Vertical Spacing no menu para fixar a distância vertical entre essas duas visões em seu valor atual. Isso é o mesmo que abrir o menu Pin e selecionar o strut superior.

Pressione a tecla Control e arraste a restrição para o rótulo Name novamente. Desta vez, selecione Left no menu. Isso é o mesmo que abrir o menu Align e marcar Leading Edges.

Finalmente, você precisará fixar a altura e a largura do rótulo em seus valores atuais. Como essas restrições afetam apenas o rótulo Serial, não pressione a tecla Control e arraste para outra visão. Em vez disso, faça um movimento bem curto na diagonal pressionando a tecla Control e arrastando a restrição dentro do rótulo Serial.

Quando o menu aparecer, mantenha pressionada a tecla Shift para selecionar tanto Width quanto Height. (Se você está vendo apenas uma dessas opções, então seu movimento de pressionar Control e arrastar foi feito na vertical ou horizontal em vez de diagonal. Tente novamente.)

Agora você precisa restringir o campo de texto à direita do rótulo **Serial**. Essa visão precisa de três restrições:

- a borda esquerda deve ser fixada à distância atual do rótulo **Serial**
- a linha de base deve ser alinhada à linha de base do rótulo **Serial**
- a borda direita deve ser fixada à distância atual de seu recipiente

Selecione o campo de texto, pressione a tecla Control e arraste a restrição para o rótulo **Serial**. Solte e clique em **Horizontal Spacing** e **Baseline** pressionando Shift. Em seguida, pressione Control e arraste a restrição do campo de texto à direita para a supervisão. Selecione **Trailing Space to Container**.

Existem mais três visões que precisam de restrição: o rótulo **Value**, o campo de texto à direita e o rótulo que exibe a data. Usando o menu de restrições ou pressionando Control e arrastando, adicione as seguintes restrições:

No caso do rótulo **Value**...

- a borda superior deve ser fixada à sua distância atual do rótulo **Serial**
- a borda condutora deve estar alinhada à borda condutora do rótulo **Serial**
- a altura e a largura devem ser fixadas em seus valores atuais

No caso do campo de texto...

- a borda esquerda deve ser fixada à sua distância atual do rótulo **Value**
- a linha de base deve estar alinhada à linha de base do rótulo **Value**
- a borda direita deve ser fixada à distância atual de seu recipiente

No caso do rótulo de data...

- a borda superior deve ser fixada à sua distância atual do campo de texto que exibe o valor do item
- as bordas esquerda e direita devem ser fixadas à distância atual de seu recipiente
- a altura deve ser fixada em seu valor atual

Compile e execute o aplicativo no iPad. A classe **BNRDetailViewController** deve ficar satisfatória neste ponto.

## Prioridades

Cada restrição possui um nível de prioridade utilizado para determinar as restrições que ganham quando mais de uma restrição entra em conflito. Uma prioridade é um valor de 1 a 1000, em que 1000 é uma restrição obrigatória. Por padrão, as restrições são obrigatórias, portanto, todas as que você adicionou são obrigatórias. Isso significa que o nível de prioridade não ajudaria se você tivesse restrições em conflito. Em vez disso, o Auto Layout relataria um problema relacionado a restrições insatisfatórias. Geralmente, você encontra as restrições que estão em conflito e então remove uma delas (o culpado costuma ser um acidente óbvio) ou reduz o nível de prioridade de uma restrição para resolver o conflito, mas manter todas as restrições em execução. Você aprenderá mais sobre como depurar esse problema na próxima seção.

## Depuração de restrições

Você adicionou restrições ao **BNRDetailViewController.xib**, e isso permitirá que o Auto Layout determine os retângulos de alinhamento de cada visão na hierarquia e forneça a você o layout que deseja no iPhone e no iPad. Em razão da grande quantidade de restrições, os problemas surgem facilmente. Você pode se esquecer de uma restrição, outras podem entrar em conflito ou ainda ter restrições que entram em conflito com a maneira como aparecem no canvas.

Felizmente, existem diversas ferramentas que podem ser usadas para depurar restrições do Auto Layout. Vamos dar uma olhada em cada uma delas e ver maneiras de corrigi-las.

## Layout ambíguo

Um layout ambíguo ocorre quando há mais de uma maneira de completar um conjunto de restrições. Geralmente, isso significa que no mínimo uma restrição está faltando.

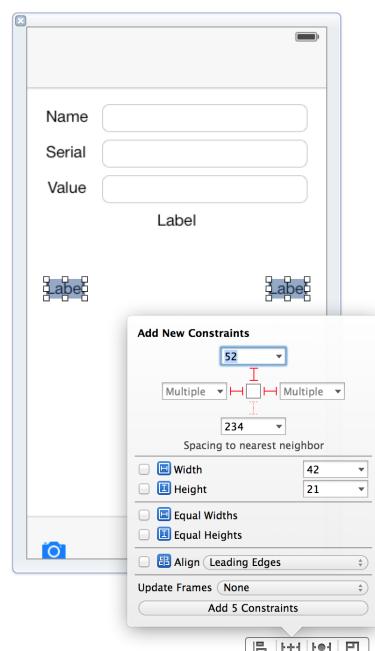
Atualmente, não há layouts ambíguos, então vou introduzir alguns. Adicione dois rótulos à **UIControl** na **dateLabel** e posicione-os um ao lado do outro. No inspetor de atributos, altere a cor de fundo dos rótulos para cinza claro para que você possa ver seus frames. A interface deve ser semelhante à da Figure 15.16.

Figure 15.16 Rótulos novos



Vamos agora adicionar algumas restrições a ambos os rótulos. Mantenha a tecla Shift pressionada e selecione ambos os rótulos. Abra o menu Pin do Auto Layout, selecione os struts superior, da esquerda e da direita no topo, depois clique em Add 5 Constraints (Figure 15.17).

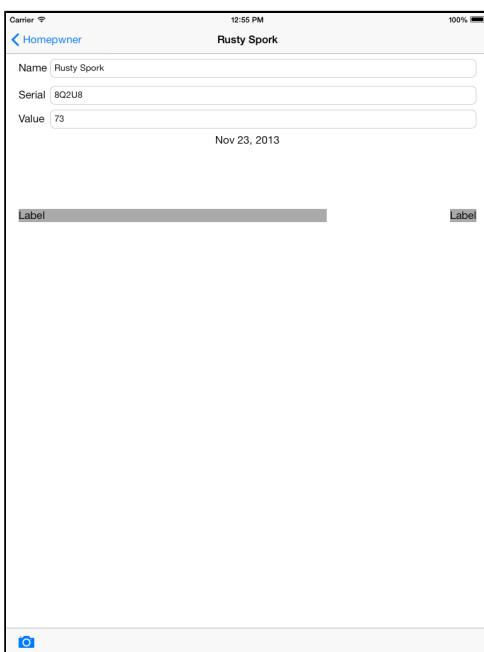
Figure 15.17 Adição de restrições a múltiplas visões ao mesmo tempo



Você fixou três bordas para dois rótulos, portanto, deve estar se perguntando: “Por que 5 restrições e não 6?”. A resposta é que a restrição da borda direita do rótulo à esquerda e a restrição da borda esquerda do rótulo à direita são as mesmas. O Interface Builder reconhece isso e adiciona apenas uma restrição para satisfazer ambos os atributos.

Se você fizer a compilação e a execução em um iPhone, tudo vai parecer normal, mas executar no iPad é uma outra história. Compile e execute o projeto em um iPad e navegue até a interface de detalhes. Um dos dois rótulos é mais extenso do que o outro (Figure 15.18).

Figure 15.18 A largura dos rótulos é surpreendente no iPad



Esses rótulos não têm restrições suficientes para definir seus frames sem ambiguidade. O Auto Layout (layout automático) faz sua melhor aposta no tempo de execução, e o resultado no iPad não é o que você esperava. Você utilizará dois métodos da **UIView**, **hasAmbiguousLayout** e **exerciseAmbiguousLayout**, para depurar essa situação.

Abra o `BNRDetailViewController.m`. Substitua o método **viewDidLoadSubviews** para verificar se alguma das subvisões possui layout ambíguo.

```
- (void)viewDidLoadSubviews
{
    for (UIView *Subview in self.view.subviews) {
        if ([Subview hasAmbiguousLayout])
            NSLog(@"AMBIGUOUS: %@", subview);
    }
}
```

**viewDidLoadSubviews** é chamado sempre que o tamanho da visão muda (por exemplo, quando o dispositivo é girado) ou quando a visão aparece pela primeira vez na tela.

Compile e execute o aplicativo no simulador de iPad e navegue até **BNRDetailViewController**. Em seguida, verifique o console, ele informará que os dois rótulos estão ambíguos. (Observe que o resultado no console costuma ficar duplicado, resultando no dobro de mensagens que você espera.)

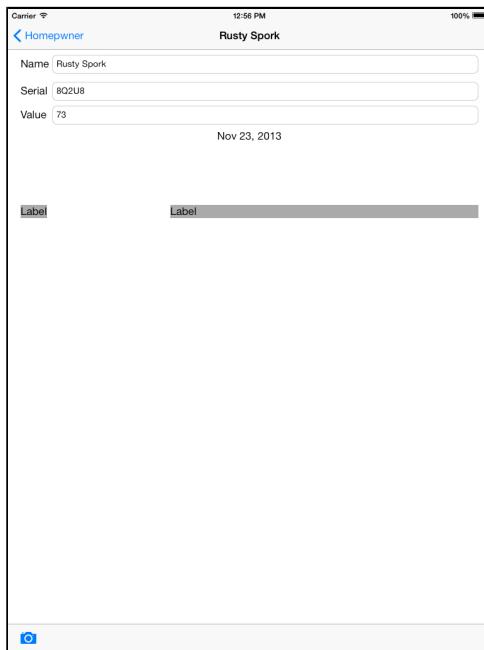
Você pode avançar uma etapa para ver a outra maneira como esse layout pode aparecer. No `BNRDetailViewController.m`, edite o método **backgroundTapped:** para enviar a mensagem **exerciseAmbiguityInLayout** para qualquer visão ambígua.

```
- (IBAction)backgroundTapped:(id)sender
{
    [self.view endEditing:YES];

    for (UIView *Subview in self.view.subviews) {
        if ([Subview hasAmbiguousLayout])
            [Subview exerciseAmbiguityInLayout];
    }
}
```

Compile e execute o aplicativo novamente. Quando estiver na **BNRDetailViewController**, toque em qualquer lugar na visão de fundo. A largura dos dois rótulos será alternada.

Figure 15.19 Tocar no fundo exibe o outro layout possível



Nenhuma das larguras dos rótulos foi restringida e, portanto, há mais de uma solução para o sistema de equações do Auto Layout. Por causa disso, há um layout ambíguo, e o toque no fundo faz as duas soluções possíveis serem alternadas. Por causa das outras restrições especificadas, desde que um dos rótulos tenha restrição de largura, a largura do outro rótulo pode ser determinada. Você se livrará do layout ambíguo definindo larguras iguais para os rótulos.

No **BNRDetailViewController.xib**, pressione Control e arraste de um rótulo para o outro, e selecione **Equal Widths**. Compile e execute o aplicativo no iPad. Os rótulos terão a mesma largura. Verifique o console para confirmar que não existem mais layouts ambíguos. O toque no fundo não afetará a interface.

Sua interface está novamente configurada de forma adequada. Todas as visões têm restrições suficientes para construir seu retângulo de alinhamento, portanto, não há mais visões ambíguas.

No **BNRDetailViewController.xib**, selecione e exclua os dois rótulos de teste.

O método **exerciseAmbiguityInLayout** é puramente uma ferramenta de depuração que permite que o Auto Layout mostre a você como seus layouts poderão ficar no final. Nunca deixe esse código em um aplicativo que você irá despachar.

No **BNRDetailViewController.m**, exclua **viewDidLayoutSubviews** e exclua o código que chama o **exerciseAmbiguityInLayout** no **backgroundTapped:**.

```
- (void)viewDidLayoutSubviews
{
    for (UIView *Subview in self.view.subviews) {
        if ([Subview hasAmbiguousLayout])
            NSLog(@"%@", Subview);
    }
}

- (IBAction)backgroundTapped:(id)sender
{
    [self.view endEditing:YES];

    for (UIView *Subview in self.view.subviews) {
        if ([Subview hasAmbiguousLayout])
            [Subview exerciseAmbiguityInLayout];
    }
}
```

## Restrições insatisfatórias

O problema das restrições insatisfatórias ocorre quando duas ou mais restrições entram em conflito. Isso costuma significar que uma visão contém restrições em excesso. Para ilustrar, vamos introduzir esse problema na `BNRDetailViewController`.

No `BNRDetailViewController.xib`, selecione o rótulo que exibe a data. No inspetor de atributos, altere o fundo para cinza claro para que você possa ver seu frame no layout. Em seguida, fixe a largura desse rótulo no valor atual.

Assim como antes, se você compilar e executar o aplicativo no iPhone, tudo vai parecer estar correto. Compile e execute o aplicativo no iPad. O rótulo vai parecer estar exatamente como antes, mas dê uma olhada no console.

```
Unable to simultaneously satisfy constraints.
Probably at least one of the constraints in the following list is one you don't want.
Try this: (1) look at each constraint and try to figure out which you don't expect;
(2) find the code that added the unwanted constraint or constraints and fix it.
(Note: If you're seeing NSAutoresizingMaskLayoutConstraints that you don't understand,
refer to the documentation for the UIView property
translatesAutoresizingMaskIntoConstraints)
(
    "<NSLayoutConstraint:0xa333da0 H:[UILabel:0xa333ca0(280)]>",
    "<NSLayoutConstraint:0xa394500 H:[UILabel:0xa333ca0]-(20)- |
        (Names: '|':UIControl:0xa38cd80 )>",
    "<NSLayoutConstraint:0xa394530 H:|- (20)-[UILabel:0xa333ca0]
        (Names: '|':UIControl:0xa38cd80 )>",
    "<NSAutoresizingMaskLayoutConstraint:0xa3a1a70 h=-&-
        v=-&- UIControl:0xa38cd80.width == _UIParallaxDimmingView:0xa37b140.width>",
    "<NSAutoresizingMaskLayoutConstraint:0xa3a21d0 h=-&
        v=-& H:_UIParallaxDimmingView:0xa37b140(768)]>"
)
Will attempt to recover by breaking constraint
<NSLayoutConstraint:0xa333da0 H:[UILabel:0xa333ca0(280)]>

Break on objc_exception_throw to catch this in the debugger.
The methods in the UIConstraintBasedLayoutDebugging category on UIView listed
in <UIKit/UIView.h> may also be helpful.
```

Primeiramente, o Xcode informa que “não é possível satisfazer as restrições simultaneamente” e dá a você algumas dicas do que você deve procurar. O console, então, lista todas as restrições que estão relacionadas ao problema. Finalmente, você é informado de que uma das restrições será ignorada, assim o rótulo terá um frame válido. Nesse caso, a restrição para fixar a largura será ignorada.

Você pode também apenas refletir sobre o problema. Você restringiu as bordas esquerda e direita desse rótulo para redimensioná-las de acordo com a supervisão. Por isso, restringiu sua largura a um valor fixo. Essas são as restrições conflitantes, e a solução é remover uma delas.

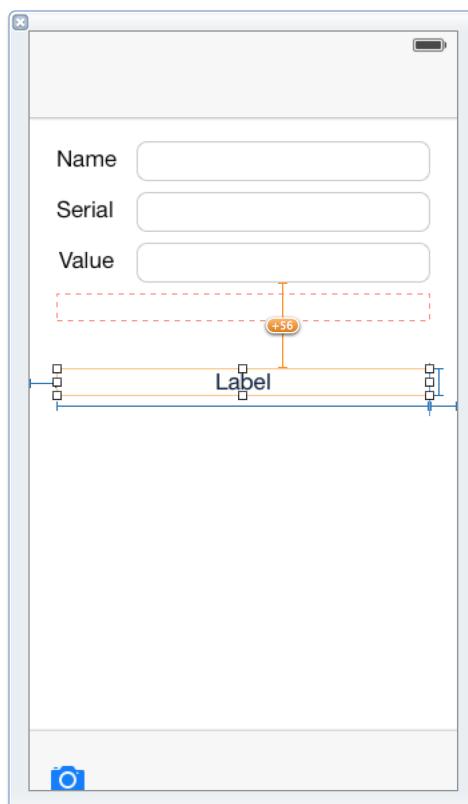
No `BNRDetailViewController.xib`, exclua a restrição de largura que acabou de adicionar ao rótulo e defina a cor de fundo de volta para transparente.

## Visões posicionadas incorretamente

Se o frame de uma visão em um XIB não corresponde às suas restrições, então surge um problema de visão posicionada incorretamente. Isso significa que o frame dessa visão no tempo de execução não corresponde a como ele está aparecendo atualmente no canvas. Vamos causar um problema de visão posicionada incorretamente.

Selecione o rótulo que exibe a data e arraste-o um pouco para baixo, assim a interface ficará como na Figure 15.20.

Figure 15.20 Visão posicionada incorretamente

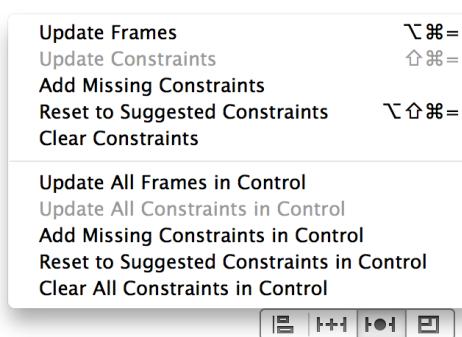


Um retângulo com borda laranja tracejada aparecerá onde o rótulo costumava ficar. Esse é o frame de tempo de execução; no tempo de execução, as restrições existentes posicionarão o rótulo no local onde esse retângulo está e não para onde você acabou de arrastá-lo.

A maneira como você corrigirá o problema dependerá do tamanho e a posição da visão estarem no canvas da forma que você deseja. Se estiverem, então altere as restrições para trabalharem com essa nova posição. Se não estiverem, então altere o tamanho ou a posição da visão, de modo que correspondam às restrições. Digamos que a movimentação do rótulo tenha sido um acidente e que você deseja que a posição da visão seja correspondente às restrições existentes.

Selecione o rótulo de data. Em seguida, no menu de restrições do Auto Layout, selecione o ícone para revelar o menu **Resolve Auto Layout Issues** (Figure 15.21).

Figure 15.21 Menu Resolve Auto Layout Issues (resolver problemas de layout automático)



Selecione **Update Frames** na parte superior. O rótulo será repositionado para que corresponda a suas restrições.

Por outro lado, se você quiser que as restrições fossem alteradas para corresponderem à nova posição da visão, você deveria escolher **Update Constraints**.

O menu Resolve Auto Layout Issues é muito poderoso. Veja uma descrição dos itens na metade superior do menu.

|                                |                                                                                                                                                                                                                                                                                                                                                                                                                                       |
|--------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Update Frames                  | Ajuste o frame da visão para que seja correspondente às restrições.                                                                                                                                                                                                                                                                                                                                                                   |
| Update Constraints             | Ajuste as restrições da visão para que sejam correspondentes ao frame.                                                                                                                                                                                                                                                                                                                                                                |
| Add Missing Constraints        | No caso de visões com layout ambíguo, serão adicionadas as restrições necessárias para que a ambiguidade seja removida. Contudo, as novas restrições podem não ficar da maneira como você deseja, por isso certifique-se de verificar-las novamente e testar essa solução.                                                                                                                                                            |
| Reset to Suggested Constraints | As restrições existentes serão removidas da visão e novas restrições serão adicionadas. Essas restrições sugeridas são sensíveis ao contexto da visão. Por exemplo, se a visão estiver próxima da parte superior da sua supervisão, as restrições sugeridas provavelmente a fixarão na parte superior, ao passo que, se a visão estiver próxima da parte inferior da sua supervisão, provavelmente ela será fixada na parte inferior. |
| Clear Constraints              | Todas as restrições serão removidas. Se nenhuma restrição explícita for adicionada a essa visão, serão adicionadas a ela restrições padrão, com tamanho e posição fixos.                                                                                                                                                                                                                                                              |

A seção inferior repete essas opções, mas as aplica a todas as subvisões, em vez de aplicá-las apenas à(s) visão(ões) selecionada(s).

## Desafio de bronze: a prática leva à perfeição

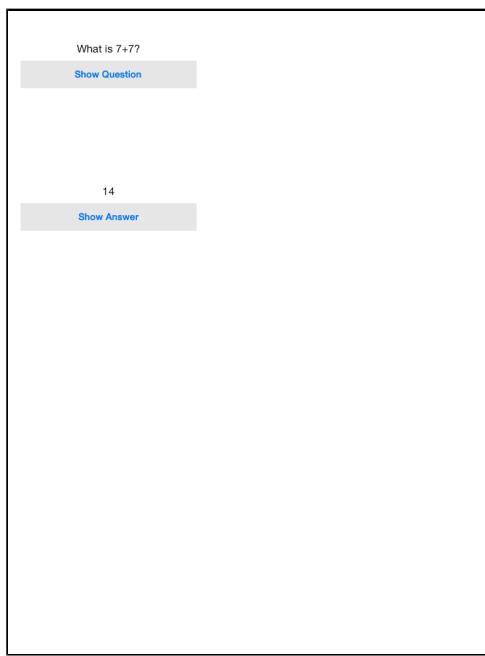
Abra o menu Resolve Auto Layout Issues e selecione Clear All Constraints in Control. Veja novamente a Figure 15.4 ou execute o aplicativo no simulador de iPad e navegue até a interface de detalhes para se lembrar dos problemas iniciais. Adicione você mesmo as restrições novamente a fim de obter uma interface de detalhes com aparência razoável no iPad.

Brinque com diferentes maneiras de adicionar restrições (menus vs. pressionar Control e arrastar), adicionando múltiplas restrições de uma só vez e restringindo múltiplas visões ao mesmo tempo. Use as ferramentas de depuração e os avisos e erros que o Interface Builder disponibiliza, para garantir que suas restrições sejam suficientes.

## Desafio de prata: universalizar o Quiz

Transforme o Quiz em um aplicativo universal. Se você executar o aplicativo universalizado no simulador de iPad sem adicionar restrições, a interface ficará desta maneira:

Figure 15.22 Quiz universalizado, sendo executado no iPad



Decida como a aparência da interface ficará no iPad e adicione restrições ao `BNRQuizViewController.xib` para se certificar de que ele será exibido da maneira desejada em qualquer dispositivo.

## Para os mais curiosos: depurar com o Auto Layout Trace (rastreamento de layout automático)

Neste capítulo, para verificar a existência de um layout ambíguo, você fez iteração por todas as subvisões da visão da `BNRDetailViewController`, e então questionou cada subvisão com `hasAmbiguousLayout`. Tudo funcionou bem porque todas as visões com as quais estava trabalhando eram subvisões da visão principal. E se sua hierarquia de visões fosse muito mais complexa? Existe outra maneira de localizar layouts ambíguos usando alguns métodos privados que a Apple revelou.

A classe `UIWindow` possui um método de instância privada denominado `_autolayoutTrace`. Isso retornará uma string com uma representação gráfica de toda a hierarquia de visões da janela e as visões com layout ambíguo serão marcadas com `AMBIGUOUS LAYOUT`.

A melhor maneira de usar isso é colocar um ponto de interrupção em algum lugar de seu código, o qual será disparado assim que a visão afetada aparecer na tela. Assim que esse ponto de interrupção for atingido, digite o código a seguir no depurador e pressione Enter.

```
(lldb) po [[UIWindow keyWindow] _autolayoutTrace]
```

Essa informação pode ser bastante útil quando sua IU não fica com a aparência esperada e você não tem certeza sobre a origem do problema.

## Para os mais curiosos: múltiplos arquivos XIB

É possível que você tenha um controlador de visão que precisa de visões completamente diferentes de acordo com o tipo de dispositivo no qual o aplicativo está sendo executado. Se esse for o caso, você pode criar dois arquivos XIB – um para cada tipo de dispositivo.

Para que o controlador de visões carregue o arquivo XIB apropriado em cada dispositivo, basta adicionar um sufixo no nome do arquivo:

```
BNRDetailViewController~iphone.xib  
BNRDetailViewController~ipad.xib
```

Ao nomear os arquivos XIB dessa maneira, o controlador de visões automaticamente encontrará e carregará o arquivo correto no tempo de execução.

Observe que usar arquivos XIB distintos não é uma alternativa ao uso do Auto Layout. Você ainda precisará adicionar restrições a ambos os arquivos. O Auto Layout também permite que sua interface responda apropriadamente a outras diferenças, como o idioma do usuário, o tamanho de fonte preferido ou a orientação vertical/horizontal do dispositivo.

# 16

## Auto Layout: restrições programáticas

Neste capítulo, você vai interagir com o Auto Layout (layout automático) por meio de código. A Apple recomenda que você crie e restrinja suas visões em um arquivo XIB sempre que possível. Contudo, se suas visões tiverem sido criadas em código, você precisará restringi-las programaticamente.

Para ter uma visão com a qual trabalhar, você recriará a visão de imagem programaticamente e a restringirá na **UIViewController**, método **viewDidLoad**. Esse método será chamado assim que o arquivo NIB da interface de **BNRDetailViewController** for carregado.

Lembre-se que no Chapter 6, você sobrescreveu **loadView** para criar visões programaticamente. Se você estiver criando e restringindo toda uma hierarquia de visões, então sobrescreva **loadView**. Se você estiver criando e restringindo uma visão adicional que será adicionada a uma hierarquia de visões criada pelo carregamento de um arquivo NIB, então sobrescreva **viewDidLoad**, neste caso.

No **BNRDetailViewController.m**, implemente **viewDidLoad** para criar uma instância de **UIImageView**.

```
- (void)viewDidLoad
{
    [super viewDidLoad];

    UIImageView *iv = [[UIImageView alloc] initWithImage:nil];
    // The contentMode of the image view in the XIB was Aspect Fit:
    iv.contentMode = UIViewContentModeScaleAspectFit;

    // Do not produce a translated constraint for this view
    iv.translatesAutoresizingMaskIntoConstraints = NO;

    // The image view was a subview of the view
    [self.view addSubview:iv];

    // The image view was pointed to by the imageView property
    self.imageView = iv;
}
```

A linha de código relacionada à tradução das restrições tem a ver com um sistema antigo de redimensionamento de interfaces: as máscaras de redimensionamento automático. Antes de o Auto Layout ser apresentado, os aplicativos para iOS utilizavam máscaras de redimensionamento automático para permitir que as visões fossem redimensionadas em telas de diferentes tamanhos no tempo de execução.

Todas as visões têm uma máscara de redimensionamento automático. Por padrão, o iOS cria restrições correspondentes à máscara de redimensionamento automático e as adiciona à visão. Essas restrições traduzidas geralmente entram em conflito com restrições explícitas no layout e causam um problema de restrições insatisfatórias. Para corrigir, desative essa tradução padrão definindo a propriedade **translatesAutoresizingMaskIntoConstraints** para **NO**. (Há mais informações sobre Auto Layout e máscaras de redimensionamento automático no final deste capítulo.)

Agora, vamos pensar em como queremos que a visão da imagem apareça. A **UIImageView** deverá ocupar toda a largura da tela e manter o espaçamento padrão de 8 pontos entre si e a **dateLabel** acima e a barra de ferramentas abaixo. Veja as restrições para a visão de imagem explicadas em detalhes:

- a borda esquerda fica a 0 ponto do recipiente da visão de imagem
- a borda direita fica a 0 ponto do recipiente da visão de imagem
- a borda superior fica a 8 pontos do rótulo de data
- a borda inferior fica a 8 pontos da barra de ferramentas

A Apple recomenda o uso de uma sintaxe especial chamada *Visual Format Language* (VFL) para criar restrições de forma programática. Essa é a maneira como você restringirá a visão de imagem. Contudo, às vezes, determinada restrição não pode ser descrita por meio da VFL. Nesses casos, deve-se adotar outra abordagem. Você verá como fazer isso no final do capítulo.

## Visual Format Language

A Visual Format Language é uma maneira de descrever restrições em uma string literal. Você pode descrever múltiplas restrições em uma única string de formatação visual. Uma única string de formatação visual, contudo, não consegue descrever restrições verticais e horizontais. Portanto, para a visão de imagem, você terá que criar duas strings de formatação visual: uma que restrinja o espaçamento horizontal da visão de imagem e outra que restrinja o espaçamento vertical.

Veja como você poderia descrever restrições de espaçamento horizontal para a visão de imagem na forma de string de formatação visual:

```
@"H: | -0-[imageView]-0- |"
```

O H: especifica que essas restrições se referem ao espaçamento horizontal. A visão é identificada dentro dos colchetes. A barra vertical (|) significa o recipiente da visão. Essa visão de imagem, portanto, ficará 0 ponto de distância de seu recipiente nas bordas esquerda e direita.

Quando o número de pontos entre a visão e o recipiente (ou alguma outra visão) é 0, os traços e o 0 podem ficar de fora da string:

```
@"H: | [imageView] |"
```

A string para as restrições verticais é semelhante à seguinte:

```
@"V: [dateLabel]-8-[imageView]-8-[toolbar]"
```

Observe que a parte “superior” e “inferior” são mapeadas à “esquerda” e à “direita”, respectivamente, nessa exibição necessariamente horizontal de espaçamento vertical. A visão de imagem fica a 8 pontos do rótulo de data na borda superior e 8 pontos da barra de ferramentas na borda inferior.

Você poderia escrever essa mesma string da seguinte maneira:

```
@"V: [dateLabel]-[imageView]-[toolbar]"
```

O traço por si só define o espaçamento para o número padrão de pontos entre as visões, que é 8.

Para saber um pouquinho mais da gramática da VFL, considere uma situação hipotética. Suponhamos que você tenha duas visões de imagem com as seguintes restrições horizontais:

- o espaçamento horizontal entre as visões de imagem dever ser de 10 pontos
- a borda esquerda da visão de imagem à esquerda ficar a 20 pontos de sua supervisão
- a borda direita da visão de imagem à direita ficar a 20 pontos de sua supervisão

Você poderia descrever as três restrições em uma única string de formatação visual:

```
@"H: | -20-[imageViewLeft]-10-[imageViewRight]-20- |"
```

A sintaxe de uma restrição de tamanho fixo é simplesmente adicionar um operador de igualdade e um valor entre parênteses dentro da formatação visual da visão:

```
@"V:[someView(==50)]"
```

A altura da visão seria restringida em 50 pontos.

## Criação de restrições

Uma restrição é uma instância da classe **NSLayoutConstraint**. Ao criar restrições de forma programática, você cria explicitamente uma ou mais instâncias de **NSLayoutConstraint** e as adiciona ao objeto de visão apropriado. Criar e adicionar restrições são uma das etapas ao se trabalhar com um XIB, porém são sempre duas etapas distintas em código.

Você cria restrições a partir de uma string de formatação visual usando o método **NSLayoutConstraint**:

```
+ (NSArray *)constraintsWithVisualFormat:(NSString *)format
                                    options:(NSLayoutFormatOptions)opts
                                    metrics:(NSDictionary *)metrics
                                   views:(NSDictionary *)views
```

Esse método retorna um array de objetos **NSLayoutConstraint** porque uma string de formatação visual, geralmente, cria mais de uma restrição.

O primeiro argumento é a string de formatação visual. Por enquanto, você pode ignorar os próximos dois argumentos, mas o quarto é essencial.

O quarto argumento é uma **NSDictionary** que mapeia os nomes na string de formatação visual para exibir objetos na hierarquia de visões. As duas strings de formatação visual que você utilizará para restringir a visão de imagem referem-se a objetos de visão pelos nomes das variáveis que apontam para eles.

```
@"H:|-[_imageView]-|"
@"V:[dateLabel]-[_imageView]-[toolbar]"
```

No entanto, uma string de formatação visual é apenas uma string, então, colocar o nome de uma variável dentro dela não significa nada, a menos que você faça a associação de forma explícita.

No `BNRDetailViewController.m`, crie um dicionário de nomes para as visões às quais as strings de formatação visual terão de se referir.

```
...
[self.view addSubview:iv];
self.imageView = iv;

NSDictionary *nameMap = @{@"imageView" : self.imageView,
                         @"dateLabel" : self.dateLabel,
                         @"toolbar" : self.toolbar};

}
```

Você está usando os nomes de suas variáveis como chaves, mas pode usar qualquer chave para nomear uma visão. A única exceção é o caractere `|`, que é um nome reservado para a supervisão (recipiente) das visões que estão sendo mencionadas na string.

Em seguida, no `BNRDetailViewController.m`, crie as restrições horizontais e verticais para a visão de imagem:

```
- (void)viewDidLoad
{
    [super viewDidLoad];
    ...
    NSDictionary *nameMap = @{@"imageView" : self.imageView,
                               @"dateLabel" : self.dateLabel,
                               @"toolbar" : self.toolbar};

    // imageView is 0 pts from superview at left and right edges
    NSArray *horizontalConstraints =
        [NSLayoutConstraint constraintsWithVisualFormat:@"H:|-0-[imageView]-0-|"
                                                options:0
                                                metrics:nil
                                                views:nameMap];

    // imageView is 8 pts from dateLabel at its top edge...
    // ... and 8 pts from toolbar at its bottom edge
    NSArray *verticalConstraints =
        [NSLayoutConstraint constraintsWithVisualFormat:
            @"V:[dateLabel]-[imageView]-[toolbar]"
            options:0
            metrics:nil
            views:nameMap];
}
```

## Adição de restrições

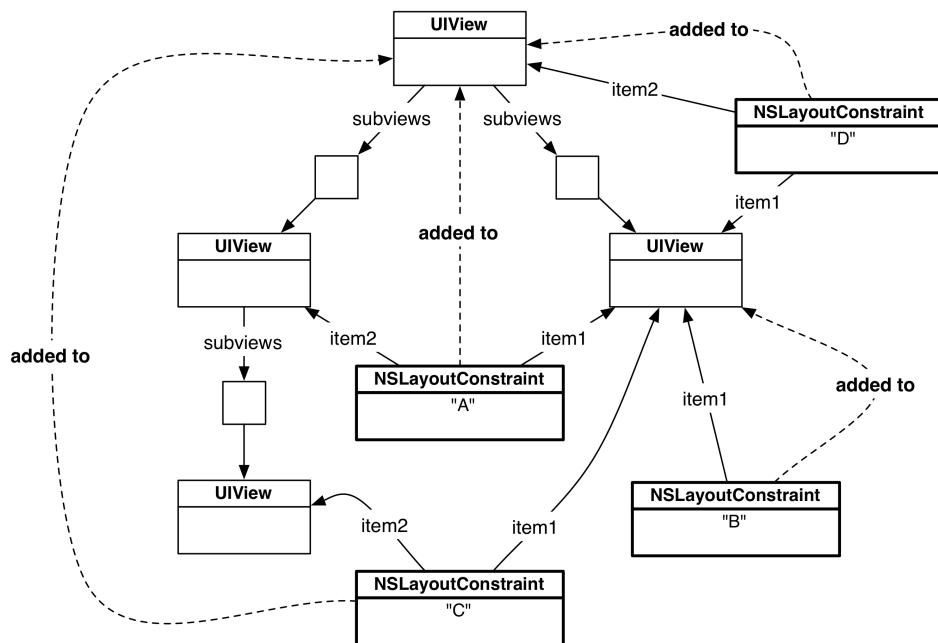
Agora você possui dois arrays de objetos **NSLayoutConstraint**. Porém, essas restrições não afetarão o layout até que você adicione-as de forma explícita por meio da **UIView**, método

```
- (void)addConstraints:(NSArray *)constraints
```

Que visão receberá a mensagem de **addConstraints:**? Geralmente, o ascendente comum mais próximo das visões que são afetadas pela restrição. Veja uma lista de regras a serem seguidas para se determinar a visão à qual você deve adicionar restrições:

- Se uma restrição afetar duas visões que têm a mesma supervisão (como a restrição denominada “A” na Figure 16.1), então a restrição deverá ser adicionada à sua supervisão.
- Se uma restrição afetar apenas uma visão (a restrição denominada “B”), então a restrição deverá ser adicionada à visão que está sendo afetada.
- Se determinada restrição afetar duas visões que não têm a mesma supervisão, mas compartilham um ascendente comum muito mais alto na hierarquia de visões (a restrição denominada “C”), então o primeiro ascendente comum recebe a restrição.
- Se determinada restrição afetar uma visão e sua supervisão (a restrição denominada “D”), então a restrição será adicionada à supervisão.

Figure 16.1 Hierarquia de restrições



Para as restrições horizontais da visão de imagem, esta determinação é fácil. Essas restrições afetam apenas a `imageView` e sua supervisão, portanto, adicione-as à supervisão – a view de `BNRDetailViewController`.

No caso de restrições verticais, a `imageView`, `dateLabel` e `toolbar` são as visões afetadas. Todas elas compartilham a mesma supervisão (a visão da `BNRDetailViewController`), por isso adicione também essas restrições à supervisão.

No `BNRDetailViewController.m`, adicione ambos os conjuntos de restrições à view de `BNRDetailViewController`:

```

...
NSArray *verticalConstraints =
    [NSLayoutConstraint constraintsWithVisualFormat:
        @"V:[dateLabel]-[imageView]-[toolbar]"
        options:0
        metrics:nil
        views:nameMap];

[self.view addConstraints:horizontalConstraints];
[self.view addConstraints:verticalConstraints];
}

```

Compile e execute o aplicativo. Crie um item e selecione uma imagem. Sua interface de detalhes pode parecer satisfatória ou não. Tudo dependerá do tamanho da imagem que você selecionar. Se a imagem selecionada for pequena, você talvez visualize algo assim:

Figure 16.2 Imagem pequena com resultados inesperados



Para entender o que está ocorrendo, vamos ver o tamanho do conteúdo intrínseco da visão e como ele interage com o Auto Layout.

(Observação: Se a `valueField` estiver ausente, isso significa que há um bug no Auto Layout. Restrições de linha de base, como aquelas entre `valueField` e o rótulo `Value`, não estão sendo respeitadas. Remova essa restrição de linha de base e substitua-a por uma restrição `CenterY`. Se você usar a abordagem "pressionar Control e arrastar" para criar essa restrição, a constante para a atual diferença entre centros será definida, mantendo de fato o alinhamento de linha de base.)

## Tamanho de conteúdo intrínseco

Tamanho de conteúdo intrínseco trata das informações que uma visão possui em relação ao tamanho que ela deve ter com base naquilo que ela exibe. Por exemplo, o tamanho de conteúdo intrínseco de um rótulo baseia-se na quantidade de texto que está sendo exibido. Em seu caso, o tamanho de conteúdo intrínseco da visão de imagem é o tamanho da imagem que você selecionou.

O Auto Layout leva essas informações em consideração ao criar restrições com tamanho de conteúdo intrínseco para cada visão. Diferentemente de outras restrições, essas têm duas prioridades: uma prioridade de envolvimento de conteúdo e uma prioridade de resistência à compressão de conteúdo.

A

prioridade de envolvimento de conteúdo

informa ao Auto Layout a importância de o tamanho da visão ficar próxima a, ou “envolver”, o conteúdo intrínseco. Um valor de 1000 significa que a visão não deve nunca ficar maior que seu tamanho de conteúdo intrínseco. Se o valor for menor que 1000, então o Auto Layout poderá aumentar o tamanho da visão, se necessário.

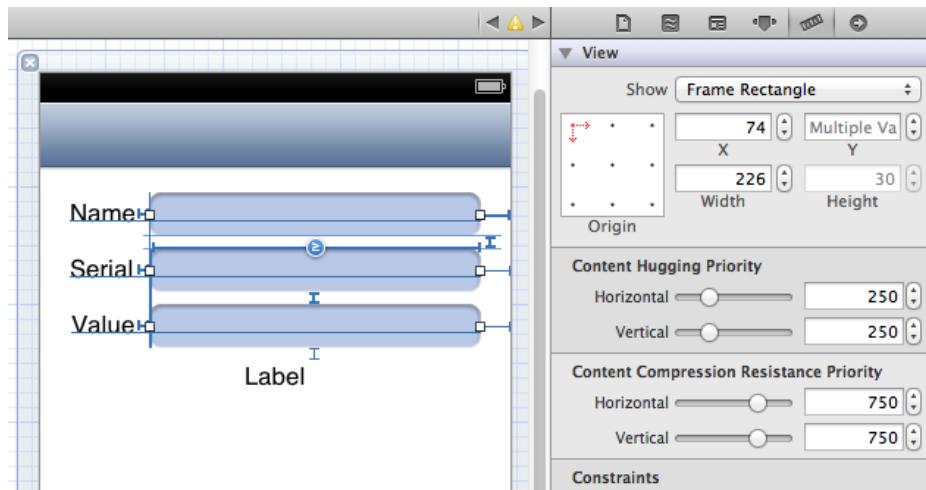
prioridade de resistência à compressão de conteúdo

informa ao Auto Layout a importância de a visão evitar o encolhimento, ou “resistir à compressão”, de seu conteúdo intrínseco. Um valor de 1000 significa que a visão não deve nunca ficar menor que seu tamanho de conteúdo intrínseco. Se o valor for menor que 1000, então o Auto Layout poderá reduzir o tamanho da visão, se necessário.

Além disso, ambas as prioridades têm valores horizontais e verticais separados, portanto, você pode definir diferentes prioridades para a altura e a largura de uma visão. Assim, totalizam-se quatro valores de prioridade de tamanho de conteúdo intrínseco por visão.

Você pode ver e editar esses valores no Interface Builder. Abra novamente o `BNRDetailViewController.xib`. No canvas, clique com a tecla Shift pressionada nos três campos de texto para selecioná-los. Vá até o inspetor e selecione a guia  para revelar o *inspetor de tamanho*. Localize as seções Content Hugging Priority e Content Compression Resistance Priority.

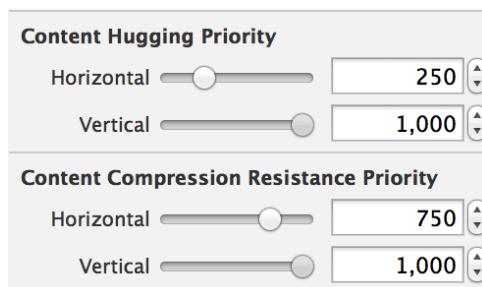
Figure 16.3 Prioridades de conteúdo



Primeiramente, observe que esses valores não são iguais a 1000 e, portanto, nunca entrarão em conflito com as restrições que você adicionou até agora. É por esse motivo que o layout aparecerá de forma incorreta, com imagens em tamanho menor. O valor da propriedade vertical de envolvimento de conteúdo do campo de texto está abaixo de 250, portanto, ao se deparar com uma imagem pequena, o Auto Layout escolhe tornar o campo de texto mais alto que seu tamanho de conteúdo intrínseco.

Seria melhor insistir que os três campos de texto fiquem com seu tamanho de conteúdo intrínseco vertical, nem maior nem menor do que seu conteúdo exige. Para isso, altere para 1000 os valores de Vertical da prioridade de envolvimento de conteúdo e da prioridade de resistência à compressão de conteúdo.

Figure 16.4 Alteração de valores verticais para 1000



Compile e execute novamente. Agora, ao lidar com imagens de tamanho menor, o Auto Layout mudará o tamanho da imagem e deixará a altura dos campos de texto inalterada.

## A outra maneira

Existem momentos em que uma restrição não pode ser criada com uma string de formatação visual. Por exemplo, você não pode usar a VFL para criar uma restrição baseada em uma proporção, por exemplo, se você quisesse que o rótulo da data tivesse o dobro da altura do rótulo do nome ou se você quisesse que a visão de imagem fosse sempre 1,5 vez mais larga do que a altura.

Nesses casos, você pode criar uma instância de `NSLayoutConstraint` usando o método

```
+ (id)constraintWithItem:(id)view1
    attribute:(NSLayoutAttribute)attr1
    relatedBy:(NSLayoutRelation)relation
        toItem:(id)view2
    attribute:(NSLayoutAttribute)attr2
    multiplier:(CGFloat)multiplier
    constant:(CGFloat)c
```

Esse método cria uma restrição única usando dois atributos de layout de dois objetos de visão. O multiplicador é a chave para a criação de uma restrição baseada em uma proporção. A constante é um número fixo de pontos, assim como você utilizou em suas restrições de espaçamento.

Os atributos de layout são definidos como constantes na classe `NSLayoutConstraint`:

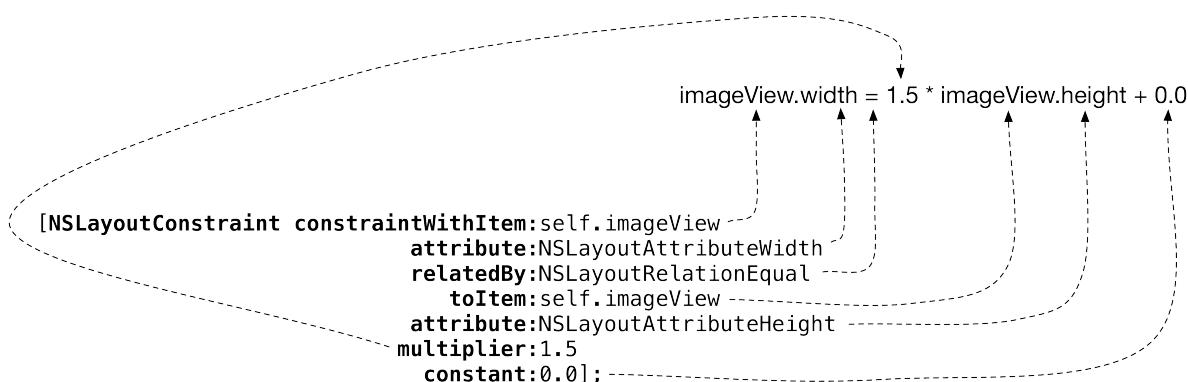
- `NSLayoutAttributeLeft`
- `NSLayoutAttributeRight`
- `NSLayoutAttributeTop`
- `NSLayoutAttributeBottom`
- `NSLayoutAttributeWidth`
- `NSLayoutAttributeHeight`
- `NSLayoutAttributeBaseline`
- `NSLayoutAttributeCenterX`
- `NSLayoutAttributeCenterY`
- `NSLayoutAttributeLeading`
- `NSLayoutAttributeTrailing`

Vamos considerar uma restrição hipotética. Digamos que você queira que a visão da imagem seja 1,5 vez mais larga do que sua altura. Você não pode fazer isso com uma string de formatação visual, pois você a criaria individualmente em vez de com o código a seguir. (Não digite esta restrição hipotética em seu código! Ela entrará em conflito com outras que você já possui.)

```
NSLayoutConstraint *aspectConstraint =
    [NSLayoutConstraint constraintWithItem:self.imageView
        attribute:NSLayoutAttributeWidth
        relatedBy:NSLayoutRelationEqual
            toItem:self.imageView
        attribute:NSLayoutAttributeHeight
        multiplier:1.5
        constant:0.0];
```

Para compreender como esse método funciona, pense nessa restrição como a equação mostrada na Figure 16.5.

Figure 16.5 Equação `NSLayoutConstraint`



Relacione um atributo de layout de uma visão ao atributo de layout de outra visão usando um multiplicador e uma constante para definir uma única restrição.

Para adicionar uma única restrição a uma visão, use o método

```
- (void)addConstraint:(NSLayoutConstraint *)constraint
```

A mesma lógica se aplica na decisão sobre qual visão deverá receber essa mensagem. Ao usar esse método, fica ainda mais fácil fazer a determinação, pois os objetos da visão afetados são o primeiro e o quarto argumentos. Nesse caso, o único objeto afetado é a visão de imagem, então você adicionaria `aspectConstraint` a essa visão:

```
[self.imageView addConstraint:aspectConstraint];
```

## Para os mais curiosos: NSAutoresizingMaskLayoutConstraint

Antes do Auto Layout, os aplicativos para iOS usavam outro sistema de gerenciamento de layout: as *máscaras de redimensionamento automático*. Cada visão tinha uma máscara de redimensionamento que restringia o relacionamento entre uma visão e sua supervisão, porém, essa máscara não afetava o relacionamento entre visões irmãs.

Por padrão, as visões criam e adicionam restrições com base em sua máscara de redimensionamento automático. No entanto, essas restrições traduzidas costumam entrar em conflito com restrições explícitas em seu layout, o que resulta em um problema de restrições insatisfatórias.

Para ver isso acontecer, comente a linha em `viewDidLoad` que desativa a tradução de máscaras de redimensionamento automático.

```
...
```

```
// The contentMode of the image view in the XIB was Aspect Fit:  
iv.contentMode = UIViewContentModeScaleAspectFit;  
  
// Turn off old-school layout handling  
iv.translatesAutoresizingMaskIntoConstraints = NO;  
  
// The image view was a subview of the view  
[self.view addSubview:iv];
```

```
...
```

Agora, a visão de imagem tem uma máscara de redimensionamento que será transformada em uma restrição. Compile e execute o aplicativo e navegue até a interface de detalhes. Você não vai gostar do que verá. O console relatará o problema e a solução.

```
Unable to simultaneously satisfy constraints.  
Probably at least one of the constraints in the following list is one you don't  
want. Try this: (1) look at each constraint and try to figure out which you don't  
expect; (2) find the code that added the unwanted constraint or constraints and  
fix it. (Note: If you're seeing NSAutoresizingMaskLayoutConstraints that you don't  
understand, refer to the documentation for the UIView property  
translatesAutoresizingMaskIntoConstraints)  
  
(  
    "<NSLayoutConstraint:0x914a2e0 H:[UILabel:0x914a1e0(42)]>,"  
    "<NSLayoutConstraint:0x9153ee0  
        H:|- (20)- [UILabel:0x9149f00]   (Names: '|':UIControl:0x91496e0 )>,"  
    "<NSLayoutConstraint:0x9153fa0  
        UILabel:0x9149970.leading == UILabel:0x9149f00.leading>,"  
    "<NSLayoutConstraint:0x91540c0  
        UILabel:0x914a1e0.leading == UILabel:0x9149970.leading>,"  
    "<NSLayoutConstraint:0x9154420  
        H: [UITextField:0x914fe20]- (20)-|   (Names: '|':UIControl:0x91496e0 )>,"  
    "<NSLayoutConstraint:0x9154450  
        H: [UILabel:0x914a1e0]- (12)- [UITextField:0x914fe20]>,"  
    "<NSLayoutConstraint:0x912f5a0  
        H: |- (NSSpace(20))- [UIImageView:0x91524d0]   (Names: '|':UIControl:0x91496e0 )>,"  
    "<NSLayoutConstraint:0x91452a0  
        H: [UIImageView:0x91524d0]- (NSSpace(20))-|   (Names: '|':UIControl:0x91496e0 )>,"  
    "<NSAutoresizingMaskLayoutConstraint:0x905f130  
        h=-& v=-& UIImageView:0x91524d0.midX ==>"  
)
```

```
Will attempt to recover by breaking constraint
<NSLayoutConstraint:0x914a2e0 H:[UILabel:0x914a1e0(42)]>
```

Vamos analisar esse resultado. O Auto Layout está informando que “não é possível satisfazer restrições simultaneamente”. Isso acontece quando uma hierarquia de visões possui restrições conflitantes.

Em seguida, o console dá algumas dicas úteis e exibe uma lista com todas as restrições envolvidas. Cada **description** da restrição é mostrado no console. Vamos observar o formato de uma dessas restrições mais de perto.

```
<NSLayoutConstraint:0x9153fa0 UILabel:0x9149970.leading == UILabel:0x9149f00.leading>
```

Essa descrição indica que a restrição localizada no endereço de memória 0x9153fa0 está configurando a borda esquerda da **UILabel** (em 0x9149970) de forma igual à borda esquerda da **UILabel** (em 0x9149f00).

Quatro dessas restrições são instâncias de **NSLayoutConstraint**. A quinta, contudo, é uma instância de **NSAutoresizingMaskLayoutConstraint**. Essa restrição é o produto da transformação da máscara de redimensionamento automático da visão de imagem.

Finalmente, ela diz a você como resolverá o problema listando a restrição em conflito que ignorará. Infelizmente, ela escolhe mal e ignora uma das instâncias explícitas de **NSLayoutConstraint**, em vez de **NSAutoresizingMaskLayoutConstraint**. É por isso que sua interface ficou dessa maneira.

A observação que aparece antes de as restrições serem listadas é bastante útil: a **NSAutoresizingMaskLayoutConstraint** precisa ser removida. Ainda melhor, você pode evitar que essa restrição seja adicionada em primeiro lugar desativando de forma explícita a tradução no **viewDidLoad**:

```
...
// The contentMode of the image view in the XIB was Aspect Fit:
iv.contentMode = UIViewContentModeScaleAspectFit;

// Do not produce a translated constraint for this view
iv.translatesAutoresizingMaskIntoConstraints = NO;

// The image view was a subview of the view
[self.view addSubview:iv];
...
```

# Rotação automática, controladores popover e controladores de visão modal

Nos últimos dois capítulos, você utilizou o Auto Layout para assegurar que o Homepwner mantenha uma aparência padrão em relação ao tamanho da tela do dispositivo. Por exemplo, você fez com que a barra de ferramentas esteja sempre na parte inferior da tela e tenha a mesma largura que a tela.

Ao projetar um aplicativo universal, muitas vezes você precisa que o aplicativo se comporte de forma diferente dependendo do tipo de aparelho sendo usado. Os diferentes dispositivos têm expressões diferentes esperadas pelos usuários, de forma que comportamentos ou características iguais em todos os dispositivos podem parecer, por vezes, estranhos.

Neste capítulo, você fará quatro modificações no comportamento do Homepwner que adequarão o comportamento do aplicativo a qualquer que seja o dispositivo em que ele estiver sendo executado.

- Apenas em iPads, permitir que a interface gire quando o aparelho estiver de cabeça para baixo.
- Apenas em iPads, mostrar o seletor de imagens em um controlador popover quando o usuário pressionar o botão da câmera.
- Apenas em iPads, apresentar a interface de detalhes de forma modal quando o usuário criar um novo item.
- Apenas em iPhones, desativar o botão da câmera na interface de detalhes quando o dispositivo estiver em orientação paisagem.

Assim, você aprenderá como testar para o tipo de dispositivo e escrever código específico ao dispositivo. Você também aprenderá sobre rotação, controladores popover e mais sobre controladores de visão modais.

## Rotação automática

Há duas orientações distintas no iOS: *orientação do dispositivo* e *orientação da interface*.

A orientação do dispositivo representa a sua orientação física, se está com o lado direito para cima, de cabeça para baixo, se foi girado para a esquerda, direita, para cima ou para baixo. Você pode acessar a orientação do dispositivo através da propriedade `orientation` da classe `UIDevice`.

Em contraste, a orientação da interface é uma propriedade do aplicativo em execução. A seguinte lista mostra todas as orientações possíveis da interface:

|                                                       |                                                                   |
|-------------------------------------------------------|-------------------------------------------------------------------|
| <code>UIInterfaceOrientationPortrait</code>           | O botão Home está abaixo da tela.                                 |
| <code>UIInterfaceOrientationPortraitUpsideDown</code> | O botão Home está acima da tela.                                  |
| <code>UIInterfaceOrientationLandscapeLeft</code>      | O dispositivo está de lado e o botão Home está à direita da tela. |
| <code>UIInterfaceOrientationLandscapeRight</code>     | O dispositivo está de lado e o botão                              |

Home está à esquerda da tela.

Quando a orientação da interface do aplicativo muda, o tamanho da janela do aplicativo também muda. A janela irá assumir o novo tamanho e girar sua hierarquia de visão. As visões na hierarquia se ajustarão de acordo com suas restrições.

Abra o Homepwner.xcodeproj e compile o aplicativo no simulador do iPad.

Enquanto o Homepwner estiver sendo executado no simulador, você pode simular uma rotação. Navegue para **BNRDetailViewController**. No menu Hardware do simulador, selecione a opção Rotate Left. A janela do simulador girará, fazendo com que o seu “dispositivo” gire sua janela e conteúdo. Como você configurou suas restrições de forma correta, a interface fica excelente em todas as orientações.

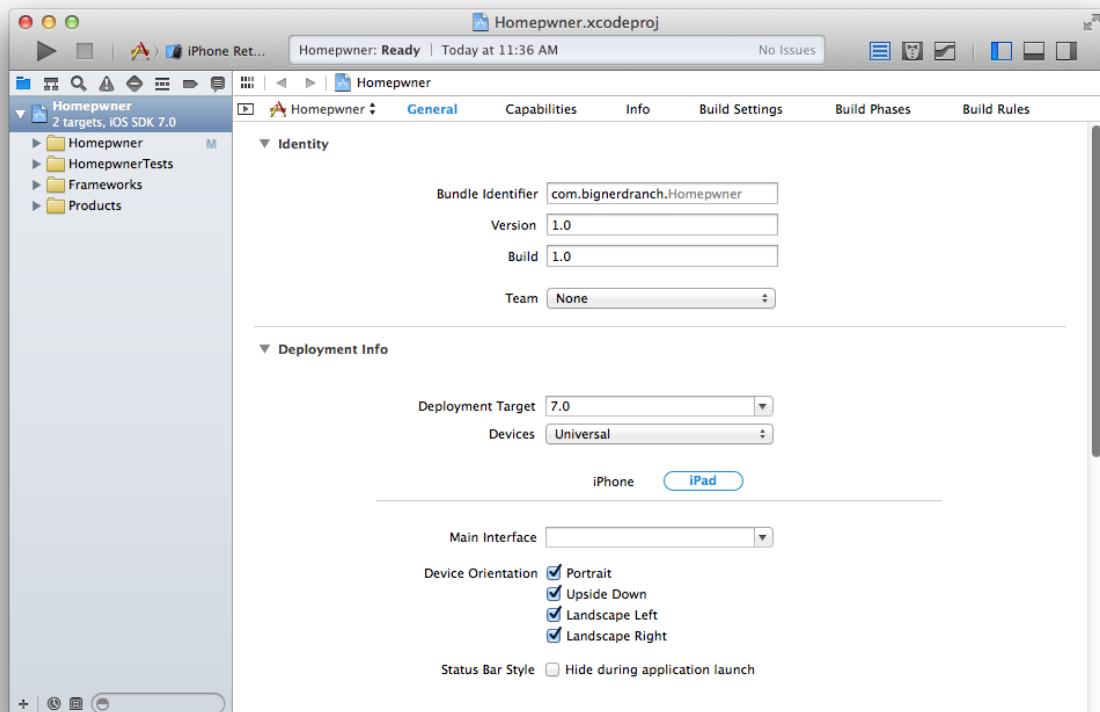
Quando a orientação do dispositivo muda, o seu aplicativo é informado sobre a nova orientação. O seu aplicativo pode decidir se irá permitir que a orientação da interface corresponda à nova orientação do dispositivo.

Gire para a esquerda outra vez para que o aplicativo fique na orientação retrato de cabeça para baixo.

Dessa vez, a interface não gira. Mesmo a orientação do dispositivo tendo mudado, a orientação da interface continuou a mesma porque o Homepwner não permite que ela seja definida como `UIInterfaceOrientationPortraitUpsideDown`. Você pode mudar as orientações de interface suportadas por um aplicativo no mesmo editor onde você universalizou o Homepwner.

Na guia General da informação Target, observe a seção Device Orientations para iPad. Veja que a opção retrato e as duas opções paisagem estão selecionadas, mas a opção Upside Down não está. Clique no botão Upside Down para ativá-la (Figure 17.1).

Figure 17.1 O Homepwner iniciado de cabeça para baixo



Compile e execute o aplicativo no simulador de iPad e gire o dispositivo na mesma direção duas vezes utilizando o menu Hardware. Agora a interface girará em todas as direções. É comum que um aplicativo de iPad possa girar em todas as quatro orientações enquanto um aplicativo de iPhone possa girar em todas as orientações excetuando-se a de cabeça para baixo. Observe que a seção do iPhone / iPod Deployment Info sustenta que esse dispositivo só pode girar para orientações retrato e paisagem enquanto está no iPhone.

Alguns aplicativos irão querer limitar o usuário para uma orientação específica. Por exemplo, vários jogos permitem apenas as duas orientações paisagem, e vários aplicativos de iPhone permitem apenas retrato. Ativar e desativar esses botões permitirá que você escolha as orientações válidas para o seu aplicativo.

Além do aplicativo escolher as orientações de interface aceitáveis, o controlador de visão que ocupa a tela também tem influência nisso. (No Homepwner, o **UINavigationController** ocupa a tela, exceto quando o **BNRDetailViewController** é apresentado como modal.) Cada controlador de visão implementa um método que retorna todas as orientações de interface suportadas. Para que a orientação da interface seja alterada, tanto o **rootViewController** do aplicativo quanto o aplicativo em si (através da seção Supported Interface Orientations da lista de propriedades de informações) devem concordar se a nova orientação é adequada.

Por padrão, um controlador de visão em execução no iPad permitirá todas as orientações. Um controlador de visão no aplicativo de iPhone permitirá todas as orientações, exceto a de cabeça para baixo. Se você gostaria de alterar isso, deve sobrescrever o **supportedInterfaceOrientations** naquele controlador de visão. A implementação padrão desse método é semelhante ao seguinte:

```
- (NSUInteger)supportedInterfaceOrientations
{
    if ([UIDevice currentDevice].userInterfaceIdiom == UIUserInterfaceIdiomPad) {
        return UIInterfaceOrientationMaskAll;
    } else {
        return UIInterfaceOrientationMaskAllButUpsideDown;
    }
}
```

Esse código verifica se o aplicativo está sendo executado em um iPad ou em um iPhone. Você obtém o dispositivo atual e então testa a propriedade `userInterfaceIdiom` dele. Os dois valores possíveis (quando da escrita deste livro) são `UIUserInterfaceIdiomPhone` e `UIUserInterfaceIdiomPad`.

Se, por algum motivo, você quiser que o controlador de visão raiz apareça em paisagem virado apenas para a esquerda ou apenas para a direita, você pode implementar o método da seguinte forma:

```
- (NSUInteger)supportedInterfaceOrientations
{
    // On all devices, return left and right
    return UIInterfaceOrientationMaskLandscapeLeft
        | UIInterfaceOrientationMaskLandscapeRight;
}
```

(Se você não estiver familiarizado com o operador binário OR (`|`) verifique a seção called “Para os mais curiosos: bit masks (máscaras de bits)” no final deste capítulo.)

Em vários aplicativos, a tela é ocupada por um **UINavigationController** ou um **UITabBarController**. O **UINavigationController** usa o **supportedInterfaceOrientations** herdado de **UIViewController**. Se você deseja que o controlador de visão sendo exibido pela classe **UINavigationController** determine a máscara de rotação automática, torne **UINavigationController** uma subclasse e a sobrescreva:

```
@implementation MyNavController
- (NSUInteger)supportedInterfaceOrientations
{
    return self.topViewController.supportedInterfaceOrientations;
}
@end
```

**UITabBarController** pergunta ao controlador de visão as orientações de interface suportadas para cada uma de suas guias e retorna a interseção: Isto é, **UITabBarController** somente suporta uma orientação se todas as guias suportarem.

## Notificação de rotação

Algumas vezes você irá querer fazer alguma coisa específica em um controlador de visão quando a orientação do dispositivo mudar. No Homepwner, um problema é que, no iPhone, a **UIImageView** no

**BNRDetailViewController** torna-se muito pequena na orientação paisagem. Faria mais sentido limitar o usuário a tirar e visualizar a imagem na orientação retrato. Para que isso aconteça, você deve esconder a visão de imagem e desativar o botão da câmera quando o aplicativo estiver na orientação paisagem.

Primeiro, você precisa de um ponteiro para o botão da câmera para que você possa enviar a ele uma mensagem para desativá-lo. Navegue para **BNRDetailViewController.m**. Depois, pressione Option e clique em **BNRDetailViewController.xib** para abri-lo no editor assistente.

Agora, pressione Control e arraste do botão da câmera na barra de ferramentas para a área de extensão de classe do **BNRDetailViewController.m** para criar um outlet de propriedade fraco chamado **cameraButton**. Isso criará e conectará uma nova propriedade:

```
@property (weak, nonatomic) IBOutlet UIBarButtonItem *cameraButton;
```

Agora, vamos voltar ao seu objetivo: ocultar a visão de imagem e desativar o botão da câmera apenas em paisagem e apenas se o dispositivo for um iPhone.

Ao escrever código para responder a uma mudança de orientação, você sobrescreve o método **willAnimateRotationToInterfaceOrientation:duration:** de **UIViewController**. A mensagem **willAnimateRotationToInterfaceOrientation:duration:** é enviada a um controlador de visão quando a orientação da interface muda com êxito. O novo valor da orientação da interface é o primeiro argumento desse método.

No **BNRDetailViewController.m**, crie um novo método chamado **prepareViewsForOrientation:** para verificar o dispositivo e, depois, a orientação da interface. Se o dispositivo for um iPhone e a nova orientação for paisagem, oculte a visão de imagem e desative o botão. Chame esse método quando a visão aparecer pela primeira vez na tela e chame-o outra vez sempre que a orientação mudar:

```
- (void)prepareViewsForOrientation:(UIInterfaceOrientation)orientation
{
    // Is it an iPad? No preparation necessary
    if ([UIDevice currentDevice].userInterfaceIdiom == UIUserInterfaceIdiomPad) {
        return;
    }

    // Is it landscape?
    if (UIInterfaceOrientationIsLandscape(orientation)) {
        self.imageView.hidden = YES;
        self.cameraButton.enabled = NO;
    } else {
        self.imageView.hidden = NO;
        self.cameraButton.enabled = YES;
    }
}

- (void)willAnimateRotationToInterfaceOrientation:
    (UIInterfaceOrientation)toInterfaceOrientation
    duration:(NSTimeInterval)duration
{
    [self prepareViewsForOrientation:toInterfaceOrientation];
}

- (void)viewWillAppear:(BOOL)animated
{
    [super viewWillAppear:animated];

    UIInterfaceOrientation io =
        [[UIApplication sharedApplication] statusBarOrientation];
    [self prepareViewsForOrientation:io];
    ...
}
```

Compile e execute o aplicativo no simulador de iPhone. No **BNRDetailViewController**, adicione uma imagem à **BNRItem** e então gire para paisagem. A imagem desaparecerá e o botão da câmera ficará cinza. Ao girar novamente para retrato, a imagem reaparecerá e o botão da câmera será ativado novamente. Se você compilar e executar o aplicativo no iPad, a visão de imagem e o botão de câmera sempre estarão disponíveis e ativados.

Quando você escreve um código que muda algo da visão (como seu `frame` ou se ele está oculto) nesse método, essas alterações são animadas. O argumento `duration` informa a duração dessa animação. Se você está fazendo alguma outra coisa com rotação que não envolva visões, ou se você apenas não quer que elas tenham alterações animadas, você pode sobreescriver o método `willRotateToInterfaceOrientation:duration:` em seu controlador de visão. Esse método fornece a você a mesma informação no mesmo momento, mas suas visões não são animadas automaticamente.

Além disso, se você quiser fazer algo após o término da rotação, pode sobreescriver o método `didRotateFromInterfaceOrientation:` em seu controlador de visão. Esse argumento do método é a orientação da interface anterior antes de ocorrer a rotação. A qualquer momento, você pode perguntar ao controlador de visão a orientação atual enviando para ele a mensagem `interfaceOrientation`.

Agora você escreveu algum código específico para iPhone. Na próxima seção, você irá adicionar outra característica específica ao dispositivo: mostrar o seletor de imagens em um popover quando executar o aplicativo em um iPad.

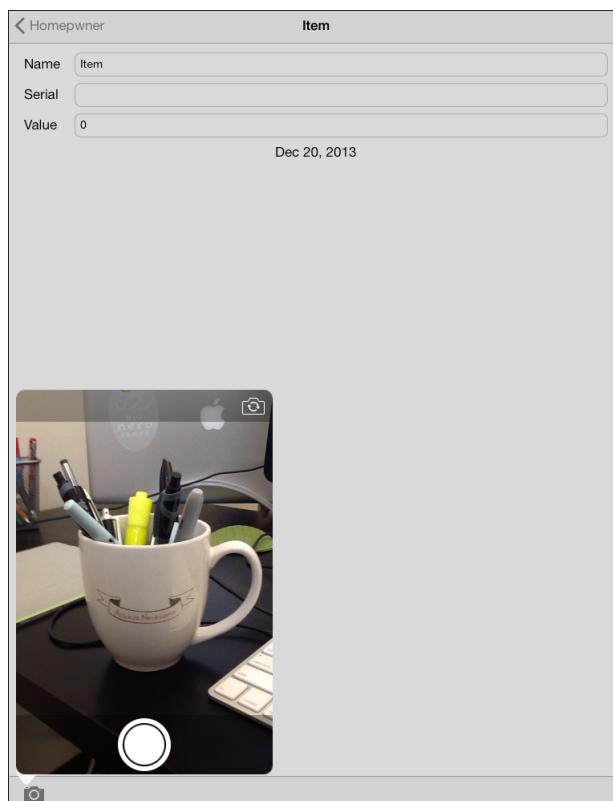
## UIPopoverController

Em aplicativos de iPad, você tem muito mais espaço de tela para trabalhar. Vamos aproveitar isso apresentando o `UIImagePickerController` em um `UIPopoverController` quando o usuário pressionar o botão de câmera na interface de detalhes.

O controlador popover mostra outra visão do controlador de visão em uma janela delimitada que flutua sobre o resto da interface do aplicativo. Ela está disponível apenas em iPads. Quando você cria um `UIPopoverController`, define esse outro controlador de visão como o `contentViewController` do controlador popover. Controladores popover são úteis ao dar ao usuário uma lista de opções (como escolher uma foto da biblioteca) ou alguma informação adicional sobre algo que está resumido na tela. Por exemplo, um formulário pode ter alguns botões ao lado de alguns dos campos. Tocar no botão revelaria um popover cujo `contentViewController` explica o uso desejado do campo.

Nesta seção, você apresentará o `UIImagePickerController` em um `UIPopoverController` quando o usuário pressionar o item de botão de barra da câmera na visão do `BNRDetailViewController` (Figure 17.2).

Figure 17.2 `UIPopoverController`



Na extensão de classe em `BNRDetailViewController.m`, declare que `BNRDetailViewController` está em conformidade com o protocolo `UIPopoverControllerDelegate`.

```
@interface BNRDetailViewController ()  
    <UINavigationControllerDelegate, UIImagePickerControllerDelegate,  
    UITextFieldDelegate, UIPopoverControllerDelegate>
```

Além disso, adicione uma propriedade para armazenar o controlador popover.

```
@interface BNRDetailViewController ()  
    <UINavigationControllerDelegate, UIImagePickerControllerDelegate,  
    UITextFieldDelegate, UIPopoverControllerDelegate>  
  
@property (strong, nonatomic) UIPopoverController *imagePickerPopover;  
@property (weak, nonatomic) IBOutlet UITextField *nameField;
```

No `BNRDetailViewController.m`, adicione o código a seguir ao final de `takePicture:`.

```
imagePickerController.delegate = self;  
  
[self presentViewController:imagePickerController animated:YES completion:nil];  
  
// Place image picker on the screen  
// Check for iPad device before instantiating the popover controller  
if ([UIDevice currentDevice].userInterfaceIdiom == UIUserInterfaceIdiomPad) {  
    // Create a new popover controller that will display the imagePickerController  
    self.imagePickerController = [[UIPopoverController alloc]  
        initWithContentViewController:imagePickerController];  
  
    self.imagePickerController.delegate = self;  
  
    // Display the popover controller; sender  
    // is the camera bar button item  
    [self.imagePickerController  
        presentPopoverFromBarButtonItem:sender  
        permittedArrowDirections:UIPopoverArrowDirectionAny  
        animated:YES];  
} else {  
    [self presentViewController:imagePickerController animated:YES completion:nil];  
}  
}
```

Observe que você verifica o dispositivo antes de criar o `UIPopoverController`. Isso é essencial. Você só pode instanciar controladores popover na família de dispositivos iPad, e tentar criar um em um iPhone resultará em uma exceção.

Compile e execute o aplicativo no simulador de iPad ou em um iPad. Navegue até `BNRDetailViewController` e toque no ícone da câmera. O popover aparecerá e mostrará a visão do `UIImagePickerController`.

Dispense o popover tocando em qualquer outro local da tela. Quando um popover é dispensado dessa forma, ele envia a mensagem `popoverControllerDidDismissPopover:` ao seu delegate.

No `BNRDetailViewController.m`, implemente o `popoverControllerDidDismissPopover:` para definir `imagePickerController` para nil para destruir o popover. Você criará um novo popover a cada vez que tocar no botão da câmera.

```
- (void)popoverControllerDidDismissPopover:(UIPopoverController *)popoverController  
{  
    NSLog(@"User dismissed popover");  
    self.imagePickerController = nil;  
}
```

O popover também será dispensado quando você selecionar uma imagem no seletor de imagens. No `BNRDetailViewController.m`, ao final do `imagePickerController:didFinishPickingMediaWithInfo:`, dispense o popover sempre que uma imagem for selecionada.

```
self.imageView.image = image;
[self dismissViewControllerAnimated:YES completion:nil];
// Do I have a popover?
if (self.imagePickerPopover) {
    // Dismiss it
    [self.imagePickerPopover dismissPopoverAnimated:YES];
    self.imagePickerPopover = nil;
} else {
    // Dismiss the modal image picker
    [self dismissViewControllerAnimated:YES completion:nil];
}
}
```

Quando você envia explicitamente a mensagem `dismissPopoverAnimated:` para dispensar o controlador popover, ele não envia o `popoverControllerDidDismissPopover:` ao seu delegate, de forma que você deve definir `imagePickerPopover` para `nil` no `dismissPopoverAnimated:` após dispensar explicitamente o popover.

Há um pequeno problema com esse código. Se `UIPopoverController` estiver visível e o usuário tocar no botão da câmera novamente, o aplicativo irá travar. Esse travamento ocorre porque o `UIPopoverController` na tela é destruído quando `imagePickerPopover` é definida para apontar para a nova classe `UIPopoverController` no `takePicture:`. Para garantir que `UIPopoverController` não fique visível e não possa ser pressionada, adicione o código a seguir no topo de `takePicture:` no `BNRDetailViewController.m`.

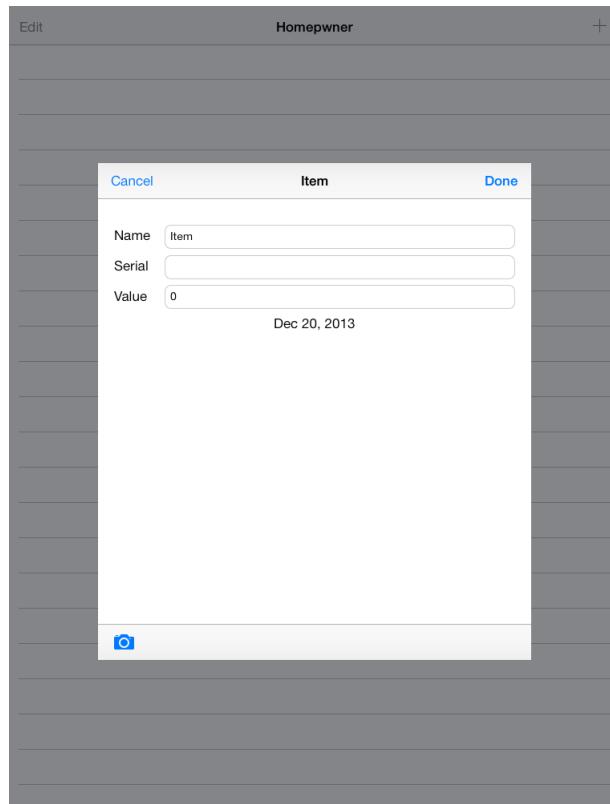
```
- (IBAction)takePicture:(id)sender
{
    if ([self.imagePickerPopover isPopoverVisible]) {
        // If the popover is already up, get rid of it
        [self.imagePickerPopover dismissPopoverAnimated:YES];
        self.imagePickerPopover = nil;
        return;
    }
    UIImagePickerController *imagePicker =
        [[UIImagePickerController alloc] init];
```

Compile e execute o aplicativo. Toque no botão da câmera para exibir o popover e depois toque novamente: o popover desaparecerá.

## Mais controladores de visão modais

Nesta parte do capítulo, você atualizará o Homepwner para apresentar a `BNRDetailViewController` como modal quando o usuário criar uma nova `BNRItem` (Figure 17.3). Quando o usuário selecionar uma `BNRItem` existente, a `BNRDetailViewController` será colocada na pilha da `UINavigationController` como antes.

Figure 17.3 Novo item



Para implementar o uso dual da **BNRDetailViewController**, você deve dar a ela um novo inicializador designado, **initForNewItem:**. Esse inicializador verificará se a instância está sendo usada para criar uma nova **BNRItem** ou exibir uma existente. Depois, ele configurará a interface adequadamente.

No **BNRDetailViewController.h**, declare esse inicializador.

```
- (instancetype)initForNewItem:(BOOL)isNew;
```

```
@property (nonatomic, strong) BNRItem *item;
```

Se **BNRDetailViewController** estiver sendo usada para criar uma nova **BNRItem**, queremos que ela mostre um botão Done e um botão Cancel em seu item de navegação. Implemente esse método no **BNRDetailViewController.m**.

```
- (instancetype)initForNewItem:(BOOL)isNew
{
    self = [super initWithNibName:nil bundle:nil];

    if (self) {
        if (isNew) {
            UIBarButtonItem *doneItem = [[UIBarButtonItem alloc]
                initWithTitle: UIBarButtonItemSystemItemDone
                target:self
                action:@selector(save:)];
            self.navigationItem.rightBarButtonItem = doneItem;

            UIBarButtonItem *cancelItem = [[UIBarButtonItem alloc]
                initWithTitle: UIBarButtonItemSystemItemCancel
                target:self
                action:@selector(cancel:)];
            self.navigationItem.leftBarButtonItem = cancelItem;
        }
    }

    return self;
}
```

Antes, quando você modificou o inicializador designado de uma classe a partir do inicializador designado da superclasse, você sobrecreveu o inicializador da superclasse para chamar o novo. Nesse caso, você vai tornar ilegal o uso do inicializador designado da superclasse disparando uma exceção sempre que ele for chamado.

No `BNRDetailViewController.m`, sobrecreva o inicializador designado da `UIViewController`.

```
- (instancetype)initWithNibName:(NSString *)NibNameOrNil  
                      bundle:(NSBundle *)NibNameOrNil  
{  
    @throw [NSError exceptionWithName:@"Wrong initializer"  
                      reason:@"Use initForNewItem:"  
                     userInfo:nil];  
    return nil;  
}
```

Esse código cria uma instância de `NSError` com um nome e um motivo, e depois dispara uma exceção. Isso faz com que o aplicativo seja interrompido e a exceção seja exibida no console.

Para confirmar se a exceção será disparada, vamos retornar para o ponto em que esse método `initWithNibName:bundle:` atualmente é chamado: o método `tableView:didSelectRowAtIndexPath:` de `BNRItemsViewController`. Nesse método, `BNRItemsViewController` cria uma instância de `BNRDetailViewController` e envia a ela a mensagem `init`, que eventualmente chama `initWithNibName:bundle:`. Portanto, a seleção de uma linha na visão de tabela resultará no disparo da exceção “Wrong initializer”.

Compile e execute o aplicativo. (Você receberá avisos de que o `save:` e o `cancel:` não estão implementados. Ignore-os por enquanto.) Toque em uma linha. Seu aplicativo será interrompido e você verá uma exceção no console. Observe que o nome e o motivo fazem parte da mensagem no console.

Você não quer mais ver essa exceção, então no `BNRItemsViewController.m`, atualize `tableView:didSelectRowAtIndexPath:` para usar o novo inicializador.

```
- (void)tableView:(UITableView *)tableView  
didSelectRowAtIndexPath:(NSIndexPath *)indexPath  
{  
    BNRDetailViewController *detailViewController =  
        [[BNRDetailViewController alloc] init];  
  
    BNRDetailViewController *detailViewController =  
        [[BNRDetailViewController alloc] initForNewItem:NO];  
  
    NSArray *items = [[BNRItemStore sharedStore] allItems];
```

Compile e execute o aplicativo novamente. Agora não acontece nada de legal nem de novo, mas o seu aplicativo não trava mais quando você seleciona uma linha na tabela.

Agora que você implementou seu novo inicializador, vamos mudar o que acontece quando o usuário adiciona um novo item.

No `BNRItemsViewController.m`, edite o método `addNewItem:` para criar uma instância de `BNRDetailViewController` em uma `UINavigationController` e apresente o controlador de navegação como modal.

```
- (IBAction)addNewItem:(id)sender
{
    BNRItem *newItem = [[BNRItemStore sharedStore] createItem];
    NSInteger lastRow = [[[BNRItemStore sharedStore] allItems] indexOfObject:newItem];
    NSIndexPath *ip = [NSIndexPath indexPathForRow:lastRow inSection:0];
    [self.tableView insertRowsAtIndexPaths:@[ip]
        withRowAnimation:UITableViewRowAnimationTop];

    BNRDetailViewController *detailViewController =
        [[BNRDetailViewController alloc] initForNewItem:YES];
    detailViewController.item = newItem;
    UINavigationController *navController = [[UINavigationController alloc]
        initWithRootViewController:detailViewController];
    [self presentViewController:navController animated:YES completion:nil];
}
```

Compile e execute o aplicativo, e toque no botão New para criar um novo item. Uma instância de **BNRDetailViewController** surgirá de baixo da tela com um botão Done e um botão Cancel em seu item de navegação. (Ao tocar nesses botões, logicamente, você irá disparar uma exceção, já que ainda não implementou os métodos de ação.)

Observe que você está criando uma instância de **UINavigationController** que nunca será usada para navegação. Isso dá a essa visão a mesma barra de título na parte superior como em todas as outras visões. Também dá a você um lugar onde colocar os botões Done e Cancel.

## Dispensando controladores de visão modais

Para dispensar um controlador de visão apresentado como modal, você deve enviar a mensagem **dismissViewControllerAnimated:completion:** para o controlador de visão que o apresentou. Você já fez isso antes com **UIImagePickerController: BNRDetailViewController** o apresentou, e quando o seletor de imagem disse para **BNRDetailViewController** que tinha terminado, **BNRDetailViewController** o dispensou.

Agora, a situação é um pouco diferente. Quando um novo item é criado, **BNRItemsViewController** apresenta **BNRDetailViewController** como modal. **BNRDetailViewController** tem dois botões em seu **navigationItem** que a dispensam quando pressionados: Cancel e Done. Há um problema aqui: as mensagens de ação para esses botões são enviadas à **BNRDetailViewController**, mas é **BNRItemsViewController** que é a encarregada de dispensar. A **BNRDetailViewController** precisa de uma maneira de dizer ao controlador de visão que a apresentou: “Ei, já terminei, você pode me dispensar agora.”

Felizmente, toda **UIViewController** tem uma propriedade **presentingViewController** que aponta para o controlador de visão que a apresentou. A **BNRDetailViewController** pegará um ponteiro para sua **presentingViewController** e enviará a ela a mensagem **dismissViewControllerAnimated:completion:**.

No **BNRDetailViewController.m**, implemente o método de ação para o botão Done.

```
- (void)save:(id)sender
{
    [self.presentingViewController dismissViewControllerAnimated:YES
                                                completion:nil];
}
```

O botão Cancel envolve algumas outras coisas. Quando o usuário toca no botão na **BNRItemsViewController** para adicionar um novo item à lista, uma nova instância de **BNRItem** é criada e adicionada ao armazenamento, e depois a **BNRDetailViewController** surge para editar este novo item. Se o usuário cancela a criação do item, essa **BNRItem** precisa ser removida do armazenamento.

No topo do **BNRDetailViewController.m**, importe o cabeçalho da **BNRItemStore**.

```
#import "BNRDetailViewController.h"
#import "BNRItem.h"
#import "BNRImageStore.h"
#import "BNRItemStore.h"

@implementation BNRDetailViewController
```

Agora, implemente o método de ação para o botão Cancel no `BNRDetailViewController.m`.

```
- (void)cancel:(id)sender
{
    // If the user cancelled, then remove the BNRItem from the store
    [[BNRItemStore sharedStore] removeItem:self.item];

    [self.presentingViewController dismissViewControllerAnimated:YES
                                                completion:nil];
}
```

Compile e execute o aplicativo. Crie um novo item e toque no botão Cancel. A instância de `BNRDetailViewController` desaparecerá da tela, e nada será adicionado à visão de tabela. Depois, crie um novo item e toque no botão Done. A `BNRDetailViewController` desaparecerá da tela e sua nova `BNRItem` aparecerá na visão de tabela.

Há uma observação final a ser feita. Dissemos que a `BNRItemsViewController` apresenta a `BNRDetailViewController` como modal. Isso é verdade em essência, mas os relacionamentos no mundo real são mais complicados que isso.

A `presentingViewController` de `BNRDetailViewController` é na verdade a `UINavigationController` que tem `BNRItemsViewController` em sua pilha. Dá para ver que esse é o caso porque quando `BNRDetailViewController` é apresentada como modal, ela cobre a barra de navegação. Se `BNRItemsViewController` estivesse manipulando a apresentação modal, a visão de `BNRDetailViewController` caberia dentro da visão de `BNRItemsViewController`, e a barra de navegação não ficaria escondida.

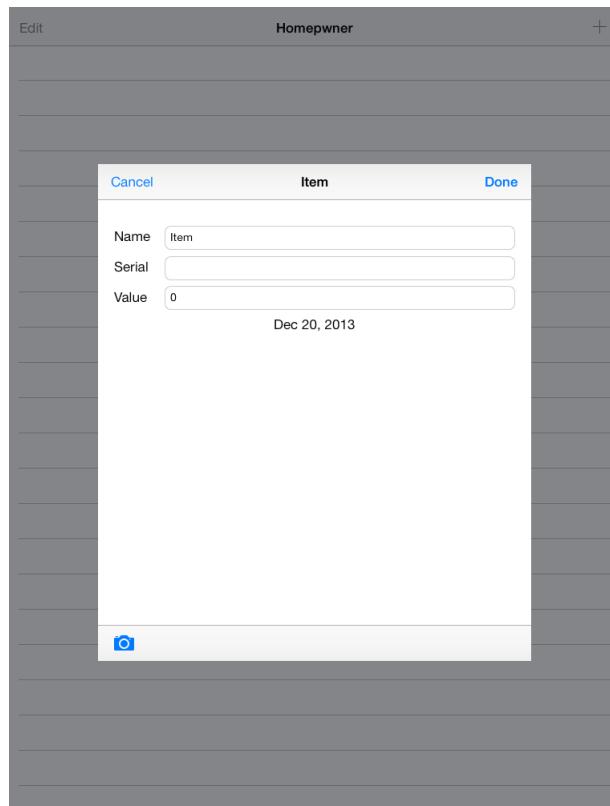
Se a finalidade for apresentar e dispensar os controladores de visão modais, isso não importa; o controlador de visão modal não se importa com quem seja sua `presentingViewController`, contanto que ele consiga enviar a ela uma mensagem e ser dispensado. Vamos abordar fatos mais complicados a respeito de relacionamentos de controlador de visão no final do capítulo.

## Estilos de controlador de visão modal

No iPhone ou iPod touch, o controlador de visão modal ocupa toda a tela. Esse é o comportamento padrão e a única possibilidade nesses dispositivos. No iPad, há duas opções adicionais: o estilo formulário e o estilo página. Você pode mudar a apresentação do controlador de visão modal definindo sua propriedade `modalPresentationStyle` para uma constante predefinida, `UIModalPresentationFormSheet` ou `UIModalPresentationPageSheet`.

O estilo formulário mostra a visão do controlador modal em um retângulo no centro da tela do iPad, e esconde a visão do controlador de visão apresentada (Figure 17.4).

Figure 17.4 Exemplo do estilo formulário



O estilo de página é igual ao estilo padrão de tela cheia no modo retrato. No modo paisagem, ele mantém a largura igual à do modo retrato, e esconde as bordas esquerda e direita da visão do controlador de visão apresentado, que aparecem por detrás dele.

No `BNRItemsViewController.m`, modifique o método `addNewItem:` para alterar o estilo de apresentação da `UINavigationController` que está sendo apresentada.

```
UINavigationController *navController = [[UINavigationController alloc]
                                         initWithRootViewController:detailViewController];
navController.modalPresentationStyle = UIModalPresentationFormSheet;
[self presentViewController:navController animated:YES completion:nil];
```

Observe que alteramos o estilo de apresentação de `UINavigationController`, e não de `BNRDetailViewController`, porque a primeira é a que está sendo apresentada como modal.

Compile e execute o aplicativo no simulador de iPad ou em um iPad. Toque no botão para adicionar um novo item e observe como o controlador de visão modal aparece na tela. Adicione alguns detalhes do item e depois toque no botão Done. A visão de tabela reaparece, mas a nova `BNRItem` não está lá. O que houve?

Antes que você alterasse o estilo de apresentação do controlador de visão modal, ele ocupava a tela inteira, o que fazia com que a visão de `BNRItemsViewController` desaparecesse. Quando o controlador de visão modal foi dispensado, `BNRItemsViewController` recebeu as mensagens `viewWillAppear:` e `viewDidAppear:` e aproveitou a oportunidade para recarregar sua tabela para capturar quaisquer atualizações da `BNRItemStore`.

Com o novo estilo de apresentação, a visão de `BNRItemsViewController` não desaparece quando ela apresenta o controlador de visão. Portanto, ela não recebe as mensagens que reaparecem quando o controlador de visão modal é dispensado, e não tem a oportunidade de recarregar sua visão de tabela.

Você precisa de outra oportunidade para recarregar os dados. O código para que `BNRItemsViewController` recarregue sua visão de tabela é simples. Isso será semelhante ao seguinte:

```
[self.tableView reloadData];
```

O que precisamos fazer é colocar o código em um pacote e executá-lo direto quando o controlador de visão modal for dispensado. Felizmente, há um mecanismo embutido no **dismissViewControllerAnimated:completion:** que você pode usar para fazer isso.

## Blocos de conclusão

Tanto no **dismissViewControllerAnimated:completion:** quanto no **presentViewController:animated:completion:**, você estava passando `nil` como o último argumento. Veja o tipo desse argumento na declaração para **dismissViewControllerAnimated:completion:**.

```
- (void)dismissViewControllerAnimated:(BOOL)flag
                           completion:(void (^)(void))completion;
```

Parece esquisito, não? Esse método espera um *bloco* como argumento, e passar um bloco aqui seria a solução do seu problema. Precisamos, então, falar sobre blocos. Porém, pode-se levar algum tempo para se acostumar com os conceitos e a sintaxe dos blocos e, portanto, vamos apenas apresentá-los brevemente. Vamos voltar a falar de blocos conforme formos seguindo no livro, e abordaremos suas diferentes características conforme for necessário.

Um bloco é um pedaço (chunk) de código a ser executado em outro momento. Você pode colocar o código para recarregar a visão de tabela em um bloco e passá-lo ao **dismissViewControllerAnimated:completion:**. A seguir, esse código será executado logo após o controlador de visão modal ser dispensado.

No `BNRDetailViewController.h`, adicione uma nova propriedade para um ponteiro de um bloco.

```
@property (nonatomic, copy) void (^dismissBlock)(void);
```

Isso determina que `BNRDetailViewController` tem uma propriedade chamada `dismissBlock` que aponta para um bloco. Como na função C, o bloco tem um valor de retorno e uma lista de argumentos. Essas características semelhantes a funções estão incluídas na declaração de um bloco. Esse bloco em particular retorna `void` e não aceita argumentos.

Você não irá criar o objeto de bloco em `BNRDetailViewController`, entretanto. Você precisa criá-lo em `BNRItemsViewController` porque `BNRItemsViewController` é o único objeto que sabe sobre a respectiva `tableView`.

No `BNRItemsViewController.m`, crie um bloco que recarrega a tabela de `BNRItemsViewController` e passe o bloco à `BNRDetailViewController`. Faça isso no método `addNewItem:` no `BNRItemsViewController.m`.

```
- (IBAction)addNewItem:(id)sender
{
    // Create a new BNRItem and add it to the store
    BNRItem *newItem = [[BNRItemStore sharedStore] createItem];

    BNRDetailViewController *detailViewController =
        [[BNRDetailViewController alloc] initForNewItem:YES];

    detailViewController.item = newItem;

    detailViewController.dismissBlock = ^{
        [self.tableView reloadData];
    };

    UINavigationController *navController = [[UINavigationController alloc]
                                             initWithRootViewController:detailViewController];
```

Agora, quando o usuário toca em um botão para adicionar um novo item, um bloco que recarrega a tabela de `BNRItemsViewController` é criado e definido como o `dismissBlock` da `BNRDetailViewController`. `BNRDetailViewController` fica com esse bloco até que o `BNRDetailViewController` precise ser dispensado.

Nesse momento, o `BNRDetailViewController` passará esse bloco ao `dismissViewControllerAnimated:completion:`.

No `BNRDetailViewController.m`, modifique as implementações do `save:` e do `cancel:` para enviar a mensagem `dismissViewControllerAnimated:completion:` com `dismissBlock` como um argumento.

```
- (IBAction)save:(id)sender
{
    [self.presentingViewController dismissViewControllerAnimated:YES
                                                    completion:nil];
    [self.presentingViewController dismissViewControllerAnimated:YES
                                                    completion:self.dismissBlock];
}

- (IBAction)cancel:(id)sender
{
    [[BNRItemStore sharedStore] removeItem:self.item];

    [self.presentingViewController dismissViewControllerAnimated:YES
                                                    completion:nil];
    [self.presentingViewController dismissViewControllerAnimated:YES
                                                    completion:self.dismissBlock];
}
```

Compile e execute o aplicativo. Toque no botão para criar um novo item e depois toque em Done. A nova `BNRItem` aparecerá na tabela.

Novamente, não se preocupe se a sintaxe ou a ideia geral dos blocos não fizer sentido nesse momento. Aguarde até o Chapter 19, quando daremos mais detalhes.

## Transições do controlador de visão modal

Além de alterar o estilo de apresentação de um controlador de visão modal, você pode alterar a animação que o coloca na tela. Assim como para os estilos de apresentação, há uma propriedade do controlador de visão (`modalTransitionStyle`) que pode ser configurada com uma constante predefinida. Por padrão, a animação faz com que o controlador de visão modal surja da parte inferior da tela. Você também pode fazer o controlador de visão surgir gradualmente, aparecer girando, ou aparecer sob uma dobradura de página.

Os vários estilos de transição são:

|                                                   |                                                                                   |
|---------------------------------------------------|-----------------------------------------------------------------------------------|
| <code>UIModalTransitionStyleCoverVertical</code>  | desliza de baixo para cima                                                        |
| <code>UIModalTransitionStyleCrossDissolve</code>  | surge gradualmente                                                                |
| <code>UIModalTransitionStyleFlipHorizontal</code> | aparece girando com um efeito 3D                                                  |
| <code>UIModalTransitionStylePartialCurl</code>    | o controlador de visão atual é descascado, revelando o controlador de visão modal |

## Singletons seguros para threads

Até agora, só falamos de aplicativos *de thread único*. Um aplicativo de thread único usa apenas um núcleo e executa apenas uma função de cada vez. Aplicativos *multi-threaded* (multisegmentados) podem executar várias funções simultaneamente em núcleos diferentes.

No Chapter 11, você criou um singleton semelhante ao seguinte:

```
+ (instancetype)sharedStore
{
    static BNRIImageStore *sharedStore = nil;

    if (!sharedStore) {
        sharedStore = [[self alloc] initPrivate];
    }
    return sharedStore;
}

// No one should call init
- (instancetype)init
{
    @throw [NSError exceptionWithName:@"Singleton"
                                reason:@"Use +[BNRIImageStore sharedStore]"
                                userInfo:nil];
    return nil;
}

- (instancetype)initPrivate
{
    self = [super init];
    if (self) {
        _dictionary = [[NSMutableDictionary alloc] init];
    }
    return self;
}
```

A técnica de singleton é suficiente em um aplicativo de thread único. Contudo, se o seu aplicativo é multisegmentado, você pode acabar criando duas instâncias de **BNRIImageStore**. Ou, você pode devolver uma instância para ser usada antes dela ser inicializada corretamente.

Você pode criar um singleton seguro para thread utilizando a função **dispatch\_once** para garantir que o código seja executado exatamente uma vez.

Abra o **BNRIImageStore.m** e altere o seu método **sharedStore** para tornar a **BNRIImageStore** um singleton seguro para thread:

```
+ (instancetype)sharedStore
{
    static BNRIImageStore *sharedStore = nil;
    if (!sharedStore) {
        sharedStore = [[self alloc] initPrivate];
    }

    static dispatch_once_t onceToken;
    dispatch_once(&onceToken, ^{
        sharedStore = [[self alloc] initPrivate];
    });
    return sharedStore;
}
```

Compile e execute. Você não deverá ver qualquer mudança no comportamento, mas o seu método **sharedStore** agora é seguro para thread.

## Desafio de bronze: outro singleton seguro para thread

Atualize o singleton da classe **BNRItemStore** para utilizar também o **dispatch\_once()**.

## Desafio de ouro: aparência do popover

Você pode modificar a aparência de uma **UIPopoverController**. Faça isso com o popover que apresenta a **UIImagePickerController**. (Dica: verifique a propriedade **popoverBackgroundViewClass** em **UIPopoverController**.)

## Para os mais curiosos: bit masks (máscaras de bits)

Anteriormente neste capítulo, você viu o método **supportedInterfaceOrientations**, que retornou todas as orientações de interface aceitáveis para um controlador de visão. Enquanto o valor de retorno desse método

era um único `int`, este `int` era, de alguma forma, capaz de indicar todas as combinações dos quatro valores de orientação de interface possíveis. Isso é possível graças a algo chamado *bitmask* (máscara de bits).

Para entender a necessidade de uma máscara de bits, considere uma solução alternativa para um controlador de visão anunciar quais orientações de interface são suportadas por ele. Uma abordagem ingênua seria dar a cada controlador de visão quatro propriedades, como a seguir:

```
@property (nonatomic, assign) BOOL canRotateToLandscapeLeft;
@property (nonatomic, assign) BOOL canRotateToLandscapeRight;
@property (nonatomic, assign) BOOL canRotateToPortrait;
@property (nonatomic, assign) BOOL canRotateToPortraitUpsideDown;
```

Com esta abordagem, cada vez que o dispositivo girasse o controlador de visão raiz fosse verificado para ver se a orientação da interface deve entrar em conformidade, a propriedade apropriada seria verificada. Máscaras de bits existem para minimizar a quantidade de código e de armazenamento necessária para representar uma série de switches ligados e desligados em uma única variável de inteiro.

Máscaras de bits são possíveis porque os computadores gravam valores em *binário*. Os números binários são strings de 1s e 0s. Aqui estão alguns exemplos de números na base 10 (decimal; a maneira como pensamos nos números) e base 2 (binário; a maneira como os computadores pensam nos números):

```
110 = 000000012
210 = 000000102
1610 = 000100002
2710 = 000110112
3410 = 001000102
```

Quando falamos em números binários, cada dígito é chamado de *bit*. Podemos pensar em cada bit como um switch liga-desliga, em que 1 significa “liga” e 0 significa “desliga”. Quando pensamos nesses termos, podemos usar um `int` (que tem espaço para no mínimo 32 bits) como uma série de switches liga-desliga. Cada posição no número representa um switch: um valor de 1 significa verdadeiro, 0 significa falso. Basicamente, estamos enfiando um monte de `BOOLs` em um único valor.

Observe que, nos exemplos acima, números como 1, 2 e 16 (potências de 2) têm apenas um dígito 1 e o resto é composto de 0s. Números que não são potências de dois, como 27 e 34, têm vários 1s em sua representação binária. Assim, podemos utilizar números que são potências de dois para representar um único switch em uma máscara de bits. Cada um desses switches é chamado de *máscara*.

Há uma máscara para cada orientação de interface possível:

```
UIInterfaceOrientationMaskPortrait = 210 = 000000102
UIInterfaceOrientationMaskPortraitUpsideDown = 410 = 000001002
UIInterfaceOrientationMaskLandscapeRight = 810 = 000010002
UIInterfaceOrientationMaskLandscapeLeft = 1610 = 000100002
```

Podemos ativar um switch em uma máscara de bits usando o operador binário OR. Essa operação pega dois números e produz um resultado em que um bit é definido como 1 quando qualquer um dos números originais tinha um 1 na mesma posição. Quando você aplica um operador binário OR a um número com  $2^n$ , ele coloca o switch na posição  $n$ . Por exemplo, se você aplicar o operador binário OR em 1 e 16, terá o seguinte:

```
00000010 (210, UIInterfaceOrientationMaskPortrait)
| 00010000 (1610, UIInterfaceOrientationMaskLandscapeLeft)
-----
00010010 (1810, both UIInterfaceOrientationMaskPortrait
            and UIInterfaceOrientationMaskLandscapeLeft)
```

O complemento do operador binário OR é o operador binário AND (&). Quando você aplica o operador binário AND em dois números, o resultado é um número que tem um 1 em cada bit onde há um 1 na mesma posição em *ambos* os números originais.

```
00010010 (1810, Portrait and Landscape Left)
```

```
& 00010000 (1610, Landscape Left)
```

```
-----
00010000 (1610, YES)
```

```
00010010 (1810, Portrait and Landscape Left)
```

```
& 00000100 (410, Upside Down)
```

```
-----
00000000 (010, NO)
```

Já que qualquer número diferente de zero significa SIM (e zero significa NÃO), utilizamos o operador binário AND para verificar se um switch está ou não ativado. Portanto, quando uma máscara **supportedInterfaceOrientations** de um controlador de visão é marcada, o código é semelhante ao seguinte:

```
if ([viewController supportedInterfaceOrientations]
    & UIInterfaceOrientationMaskLandscapeLeft)
{
    // Allow interface orientation to change to landscape left
}
```

## Para os mais curiosos: relacionamentos de controlador de visão

Os relacionamentos entre os controladores de visão são importantes para compreender onde e como a visão de um controlador de visão aparece na tela. No geral, há dois tipos diferentes de relacionamentos entre controladores de visão: relacionamentos *pai-filho* e relacionamentos *apresentado-apresentador*. Vamos analisar cada um individualmente.

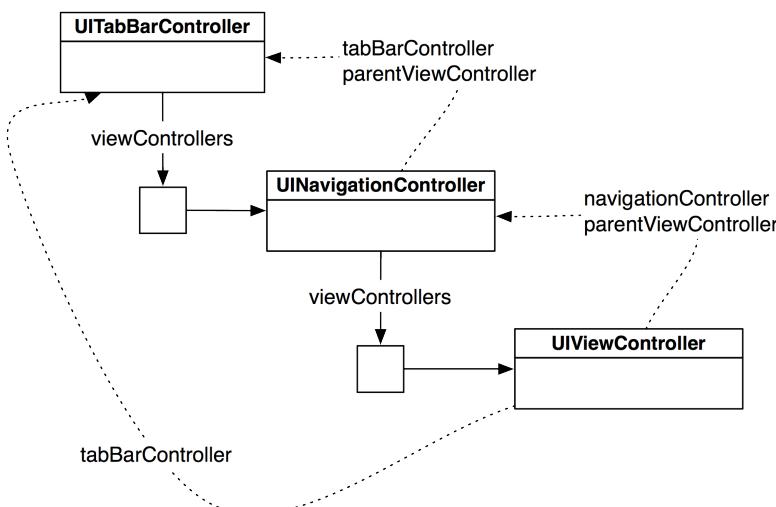
### Relacionamentos pai-filho

Os relacionamentos pai-filho são formados quando se usa *recipientes de controlador de visão*. Alguns exemplos de recipientes de controlador de visão incluem **UINavigationController**, **UITabBarController** e **UISplitViewController** (que você verá no Chapter 22). Você consegue identificar um recipiente de controlador de visão porque ele tem uma propriedade **viewControllers**, que é um array dos controladores de visão que ele contém.

Um recipiente de controlador de visão será sempre uma subclasse de **UIViewController** e, portanto, terá uma view. O comportamento de um recipiente de controlador de visão é que ele adiciona seletivamente as visões de sua **viewControllers** como subvisões de sua própria visão. O recipiente também tem sua própria interface embutida. Por exemplo, uma view de **UINavigationController** mostra uma barra de navegação e a visão de sua **topViewController**.

Os controladores de visão em um relacionamento pai-filho formam uma *família*. Assim, uma **UINavigationController** e sua **viewControllers** estão na mesma família. Uma família pode ter vários níveis. Por exemplo, imagine uma situação em que uma **UITabBarController** contém uma **UINavigationController** que contém uma **UIViewController**. Esses três controladores de visão estão na mesma família (Figure 17.5). Essas classes de recipiente têm acesso a seus filhos por meio do array **viewControllers**, e os filhos têm acesso a seus ascendentes por meio de quatro propriedades de **UIViewController**.

Figure 17.5 Uma família de controladores de visão



Cada **UIViewController** tem uma propriedade **parentViewController**. Essa propriedade armazena o controlador de visão que é o ascendente mais próximo na família. Portanto, ela poderia retornar uma

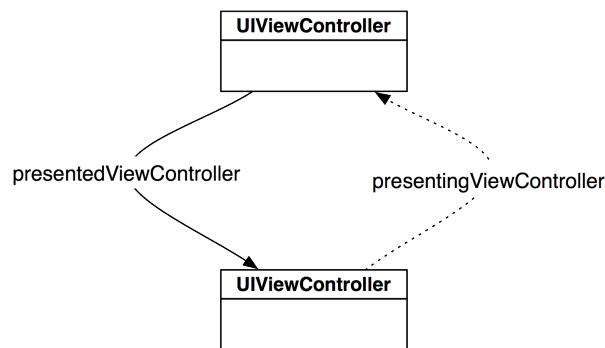
**UINavigationController**, uma **UITabBarController** ou uma **UISplitViewController** dependendo da composição da árvore genealógica.

Os métodos de acesso aos ascendentes de **UIViewController** incluem **navigationController**, **tabBarController** e **splitViewController**. Quando um controlador de visão recebe uma dessas mensagens, ele começa a percorrer a árvore genealógica (utilizando a propriedade **parentViewController**) até encontrar o tipo apropriado de recipiente de controlador de visão. Se não houver nenhum ascendente do tipo correto, esses métodos retornam **nil**.

## Relacionamentos apresentado-apresentador

O outro tipo de relacionamento é o relacionamento apresentado-apresentador, e ocorre quando um controlador de visão é apresentado como modal. Quando um controlador de visão é apresentado como modal, sua view é adicionada *sobre* a view do controlador de visão que o apresentou. Isso é diferente de um recipiente de controlador de visão, que mantém intencionalmente um espaço aberto em sua interface para acrescentar as visões dos controladores de visão que ele contém. Qualquer **UIViewController** pode apresentar outro controlador de visão como modal.

Figure 17.6 Relacionamento apresentado-apresentador



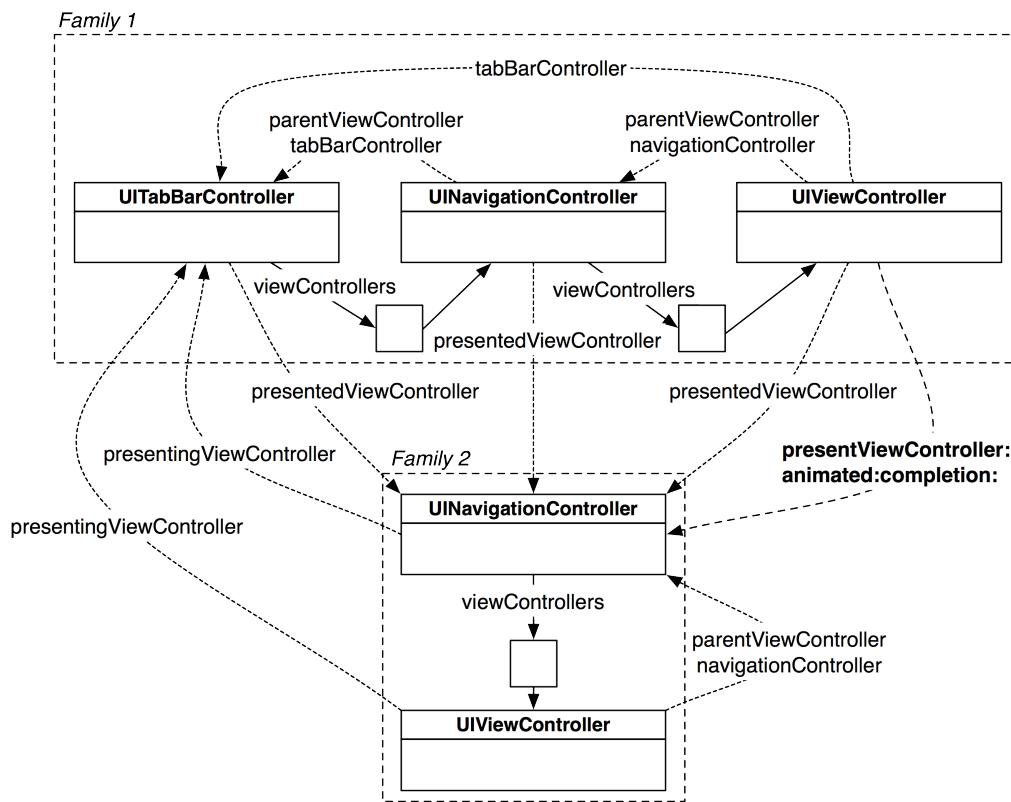
Há duas propriedades embutidas para gerenciar os relacionamentos entre o apresentador e o apresentado. Uma **presentingViewController** de um controlador de visão apresentado como modal apontará de volta para o controlador de visão que o apresentou, enquanto o apresentador manterá um ponteiro para o apresentado em sua propriedade **presentedViewController** (Figure 17.6).

## Relacionamentos interfamiliares

Um controlador de visão apresentado e seu apresentador *não* estão na mesma família de controladores de visão. Na verdade, o controlador de visão apresentado tem sua própria família. Às vezes, essa família é apenas uma **UIViewController**; em outros casos, essa família é composta por vários controladores de visão.

Compreender a diferença nas famílias ajuda a compreender os valores de propriedades como **presentedViewController** e **navigationController**. Considere os controladores de visão na Figure 17.7. Há duas famílias, cada uma com vários controladores de visão. Este diagrama mostra os valores das propriedades de relacionamento entre controladores de visão.

Figure 17.7 Uma hierarquia de controladores de visão



Primeiramente, observe que as propriedades para relacionamentos pai-filho nunca podem ultrapassar os limites das famílias. Assim, enviar o **tabBarController** para um controlador de visão na Família 2 não retornará a **UITabBarController** na Família 1; retornará `nil`. Do mesmo modo, enviar **navigationController** para o controlador de visão na Família 2 retornará o seu pai **UINavigationViewController** na Família 2, e não o **UINavigationViewController** na Família 1.

Talvez os relacionamentos mais peculiares entre controladores de visão sejam os relacionamentos entre famílias. Quando um controlador de visão é apresentado como modal, o apresentador propriamente dito é o membro mais antigo da família apresentada. Por exemplo, na Figure 17.7, a **UITabBarController** é a **presentingViewController** para os controladores de visão na Família 2. Independentemente de qual tenha sido o controlador de visão na Família 1 que recebeu o **presentViewController:animated:completion:**, a **UITabBarController** é sempre o apresentador.

Esse comportamento explica por que **BNRDetailViewController** esconde a **UINavigationBar** quando apresentada como modal, mas não quando apresentada normalmente na pilha de **UINavigationController**. Embora **BNRItemsViewController** seja instruída para apresentar como modal, seu ascendente mais antigo, **UINavigationController**, é quem realmente executa a tarefa. A **BNRDetailViewController** é colocada sobre a `view` da **UINavigationController** e, portanto, esconde a **UINavigationBar**.

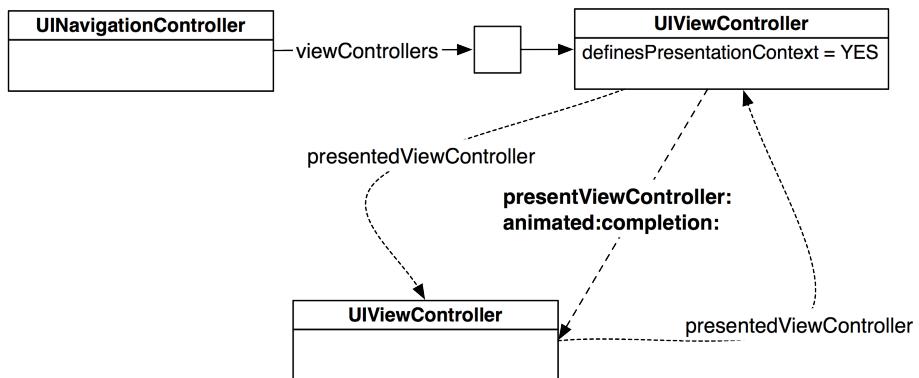
Observe também que **presentingViewController** e **presentedViewController** são válidas para todos os controladores de visão de uma família, e sempre apontam para o ascendente mais antigo na outra família.

Você pode efetivamente sobrescrever o comportamento do ascendente mais antigo (mas apenas no iPad). Ao fazer isso, você pode especificar onde as visões do controlador de visão apresentado aparecerão na tela. Por exemplo, você poderia apresentar a **BNRDetailViewController** e sua **navigationController** de forma que escondesse a **UITableView** mas não a **UINavigationBar**.

Toda **UIViewController** tem uma propriedade `definesPresentationContext` com esse objetivo. Por padrão, essa propriedade vem definida como `NO`, o que significa que o controlador de visão sempre passará a apresentação ao seu próximo ascendente, até que não reste mais nenhum ascendente. Definir essa propriedade como `YES` interrompe a busca pelo ascendente mais antigo, permitindo que o controlador de visão

apresente o controlador de visão modal em sua própria visão (Figure 17.8). Além disso, você deve definir `modalPresentationStyle` do controlador de visão apresentado para `UIModalPresentationCurrentContext`.

Figure 17.8 Contexto de apresentação



Você pode testar este recurso alterando o código no método `addNewItem:` do `BNRItemsViewController.m`.

```
UINavigationController *navController = [[UINavigationController alloc]
                                         initWithRootViewController:detailViewController];

navController.modalPresentationStyle = UIModalPresentationFormSheet;
navController.modalPresentationStyle = UIModalPresentationCurrentContext;
self.definesPresentationContext = YES;

navController.modalTransitionStyle = UIModalTransitionStyleFlipHorizontal;

[self presentViewController:navController animated:YES completion:nil];
}
```

Após compilar e executar no iPad, toque no ícone +. Observe que a `BNRDetailViewController` não esconde a `UINavigationBar`. Lembre-se de desfazer este código antes de passar para o próximo capítulo.

# Salvamento, carregamento e estados do aplicativo

Existem muitas maneiras de salvar e carregar dados em um aplicativo para iOS. Este capítulo mostrará a você alguns dos mecanismos mais comuns, bem como os conceitos necessários para fazer gravações ou leituras em um sistema de arquivos do iOS.

## Arquivamento

A maioria dos aplicativos para iOS realmente faz uma coisa: fornece uma interface para que um usuário manipule dados. Todo objeto em um aplicativo tem uma função nesse processo. Os objetos de modelo, como você já sabe, são responsáveis por manter os dados que o usuário manipula. Os objetos de visão simplesmente refletem esses dados, e os controladores são responsáveis por manter as visões e os objetos de modelo em sincronia. Portanto, ao falar sobre salvamento e carregamento de dados, estamos quase sempre falando sobre salvamento e carregamento de objetos de modelo.

No Homepwner, os objetos de modelo que um usuário manipula são instâncias de **BNRItem**. O Homepwner realmente seria um aplicativo útil se as instâncias de **BNRItem** persistissem entre as execuções do aplicativo; neste capítulo, você utilizará o *arquivamento* para salvar e carregar objetos **BNRItem**.

O arquivamento é uma das maneiras mais comuns de persistir objetos de modelo no iOS. O *arquivamento* de um objeto envolve registrar todas as suas propriedades e salvá-las no sistema de arquivos. O *desarquivamento* recria o objeto com base nesses dados.

As classes cujas instâncias precisam ser arquivadas e desarquivadas devem estar em conformidade com o protocolo **NSCoding** e implementar seus dois métodos obrigatórios, **encodeWithCoder:** e **initWithCoder:**.

```
@protocol NSCoding

- (void)encodeWithCoder:(NSCoder *)aCoder;
- (instancetype)initWithCoder:(NSCoder *)aDecoder;

@end
```

Faça com que **BNRItem** fique em conformidade com o protocolo **NSCoding**. Abra o `Homepwner.xcodeproj` e adicione a declaração desse protocolo no `BNRItem.h`.

```
@interface BNRItem : NSObject <NSCoding>
```

Agora, você precisa implementar os métodos obrigatórios. Vamos começar com o **encodeWithCoder:**. Quando uma **BNRItem** receber a mensagem **encodeWithCoder:**, ele codificará todas as suas propriedades para o objeto de **NSCoder** que é passado como um argumento. Ao salvar, você utilizará **NSCoder** para escrever um fluxo de dados. Esse fluxo será armazenado no sistema de arquivos. Esse fluxo será organizado em pares chave-valor.

No `BNRItem.m`, implemente **encodeWithCoder:** para adicionar os nomes e valores das propriedades do item ao fluxo.

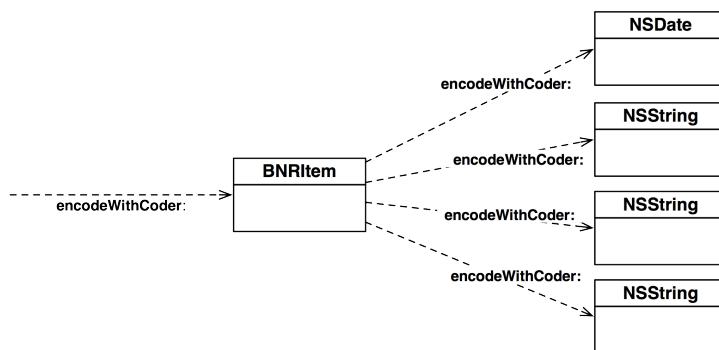
```
- (void)encodeWithCoder:(NSCoder *)aCoder
{
    [aCoder encodeObject:self.itemName forKey:@"itemName"];
    [aCoder encodeObject:self.serialNumber forKey:@"serialNumber"];
    [aCoder encodeObject:self.dateCreated forKey:@"dateCreated"];
    [aCoder encodeObject:self.itemKey forKey:@"itemKey"];
```

```
[aCoder encodeInt:self.valueInDollars forKey:@"valueInDollars"];
}
```

Observe que os ponteiros para os objetos são codificados com `encodeObject:forKey:`, mas `valueInDollars` é codificada com `encodeInt:forKey:`. Verifique a documentação sobre `NSCoder` para ver todos os tipos que você pode codificar. Independentemente do tipo do valor codificado, sempre haverá uma chave, que é uma string que identifica a variável de instância que está sendo codificada. Por convenção, essa chave é o nome da propriedade que está sendo codificada.

Quando um objeto é codificado (ou seja, é o primeiro argumento no `encodeObject:forKey:`), `encodeWithCoder:` é enviado ao objeto. Durante a execução de seu método `encodeWithCoder:`, ele codifica as variáveis de instância de seu objeto usando `encodeObject:forKey:` (Figure 18.1). Dessa forma, codificar um objeto é um processo recursivo em que cada objeto codifica seus “amigos”, que por sua vez codificam seus amigos, e assim por diante.

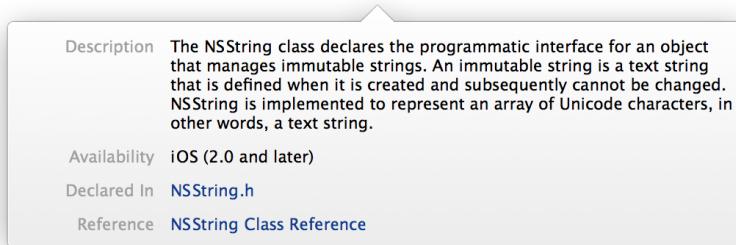
Figure 18.1 Codificação de um objeto



Para serem codificados, esses objetos devem estar em conformidade com o protocolo `NSCoding`. Vamos confirmar isso para `NSString` e `NSDate`. Os protocolos com os quais uma classe está em conformidade são listados em sua referência de classe. Em vez de abrir o navegador da documentação, como fez antes, você pode usar um atalho para acessar a referência diretamente de seu código.

No `BNRItem.m`, mantenha a tecla Option pressionada, coloque o ponteiro do mouse sobre uma ocorrência de `NSString` em seu código e clique nela. Uma janela pop-up aparecerá com uma descrição breve da classe e links para o arquivo de cabeçalho e a referência.

Figure 18.2 Clique opcional `NSString`



Clique no link para ser direcionado para a referência de classe `NSString` e, na parte superior da referência, você verá uma lista de protocolos com os quais a classe está em conformidade. O `NSCoding` não está na lista, mas se você clicar no protocolo `NSSecureCoding`, você verá que esse protocolo está em conformidade com o `NSCoding`. Portanto, `NSString` também está.

Você pode fazer a mesma verificação para a classe `NSDate` ou confiar em nós no que diz respeito a `NSDate` também estar em conformidade com `NSCoding`.

O clique opcional não é apenas para classes. Você pode usar o mesmo atalho para métodos, tipos, protocolos e muito mais. Tenha isso em mente quando se deparar com itens em seu código sobre os quais você deseja saber mais.

Agora voltaremos a falar de chaves e codificação. A finalidade da chave usada ao fazer a codificação é recuperar o valor codificado quando **BNRItem** for carregada posteriormente a partir do sistema de arquivos. Os objetos que estão sendo carregados a partir de um arquivamento recebem a mensagem **initWithCoder:**. Esse método reterá todos os objetos que foram codificados no **encodeWithCoder:** e os atribuirá à variável de instância apropriada.

No **BNRItem.m**, implemente **initWithCoder:**.

```
- (instancetype)initWithCoder:(NSCoder *)aDecoder
{
    self = [super init];
    if (self) {
        _itemName = [aDecoder decodeObjectForKey:@"itemName"];
        _serialNumber = [aDecoder decodeObjectForKey:@"serialNumber"];
        _dateCreated = [aDecoder decodeObjectForKey:@"dateCreated"];
        _itemKey = [aDecoder decodeObjectForKey:@"itemKey"];

        _valueInDollars = [aDecoder decodeIntForKey:@"valueInDollars"];
    }
    return self;
}
```

Observe que esse método também tem um argumento de  **NSCoder**. No **initWithCoder:**, a  **NSCoder** está repleta de dados a serem consumidos pela **BNRItem** que está sendo inicializada. Veja também que você enviou **decodeObjectForKey:** ao recipiente para ter os objetos de volta e enviou **decodeIntForKey:** para obter **valueInDollars**.

No Chapter 2, falamos sobre a cadeia de inicializadores e sobre os inicializadores designados. O método **initWithCoder:** não faz parte de seu padrão de projeto; você manterá o inicializador designado de **BNRItem** como está, e **initWithCoder:** não irá chamá-lo.

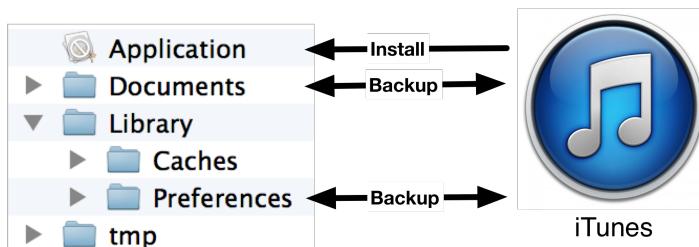
(A propósito, arquivamento é como os arquivos XIB são criados. **UIView** está em conformidade com  **NSCoding**. As instâncias de **UIView** são criadas quando você as arrasta para a área do canvas. Quando o arquivo XIB é salvo, essas visões são arquivadas no arquivo XIB. Quando seu aplicativo é iniciado, ele desarquiva as visões a partir do arquivo XIB. Existem algumas diferenças mínimas entre um arquivo XIB e um arquivo padrão, mas no geral o processo é o mesmo.)

Instâncias de **BNRItem** agora estão em conformidade com  **NSCoding** e podem ser salvas e carregadas de um sistema de arquivos por meio do arquivamento. Você pode compilar o aplicativo para certificar-se de que não há erros de sintaxe, mas você ainda precisa de um modo para iniciar os processos de salvamento e carregamento. Precisa também de um local no sistema de arquivos para armazenar os itens salvos.

## Área restrita do aplicativo

Todo aplicativo de iOS tem sua própria *área restrita do aplicativo*. Uma área restrita do aplicativo é um diretório no sistema de arquivos que é bloqueado do restante do sistema. Seu aplicativo deve ficar nessa área restrita, e nenhum outro aplicativo pode acessar essa área.

Figure 18.3 Área restrita do aplicativo



A área restrita do aplicativo contém diversos diretórios:

application bundle (pacote de aplicativos)

Esse diretório contém o executável e todos os recursos de aplicativos, como imagens e arquivos NIB.  
Ele é somente para leitura.

Documents/

Esse diretório é onde você grava os dados que o aplicativo gera durante o tempo de execução e os quais você deseja persistir entre as execuções do aplicativo. É feito o backup dele quando o dispositivo é sincronizado com o iTunes ou o iCloud. Se algo der errado com o dispositivo, os arquivos nesse diretório poderão ser restaurados a partir do iTunes ou do iCloud. Por exemplo, no Homepwner, o arquivo que contém os dados de tudo o que você possui será armazenado aqui.

Library/Caches/

Esse diretório é onde você grava os dados que o aplicativo gera durante o tempo de execução e os quais você deseja persistir entre as execuções do aplicativo. No entanto, diferente do diretório Documents, não é feito o backup dele quando o dispositivo é sincronizado com o iTunes ou o iCloud. Um motivo principal de não ocorrer o backup dos dados armazenados em cache é que os dados podem ser muito grandes e estender o tempo que se leva para sincronizar seu dispositivo. Os dados armazenados em qualquer outro local – como um servidor web – podem ser colocados nesse diretório. Se o usuário precisar restaurar o dispositivo, poderá ser feito o download desses dados a partir do servidor web novamente.

Library/Preferences/

Esse diretório é onde quaisquer preferências são armazenadas e onde o aplicativo Settings procura as preferências do aplicativo. O Library/Preferences é manipulado automaticamente pela classe **NSUserDefaults** (sobre a qual você aprenderá no Chapter 26), e seu backup é feito quando o dispositivo é sincronizado com o iTunes ou o iCloud.

tmp/

Esse diretório é onde você grava os dados que usará temporariamente durante o tempo de execução de um aplicativo. O sistema operacional pode apagar os arquivos desse diretório quando seu aplicativo não estiver sendo executado. Contudo, para fins de organização, recomenda-se ainda a remoção explícita de arquivos desse diretório quando você não precisar mais deles. Não é feito o backup desse diretório quando o dispositivo é sincronizado com o iTunes ou o iCloud. Para obter o caminho para o diretório tmp na área restrita do aplicativo, você pode usar a função de conveniência **NSTemporaryDirectory**.

## Construção de um caminho de arquivo

As instâncias de **BNRItem** do Homepwner serão salvas em um só arquivo, no diretório Documents. A **BNRItemStore** lidará com a gravação nesse arquivo e a leitura dele. Para fazer isso, a **BNRItemStore** precisa construir um caminho para esse arquivo.

Implemente um novo método no `BNRItemStore.m`.

```
- (NSString *)itemArchivePath
{
    // Make sure that the first argument is NSDocumentDirectory
    // and not NSDocumentationDirectory
    NSArray *documentDirectories =
        NSSearchPathForDirectoriesInDomains(NSDocumentDirectory,
                                            NSUserDomainMask, YES);

    // Get the one document directory from that list
    NSString *documentDirectory = [documentDirectories firstObject];

    return [documentDirectory stringByAppendingPathComponent:@"items.archive"];
}
```

A função **NSSearchPathForDirectoriesInDomains** faz uma busca no sistema de arquivos para localizar um caminho que atenda aos critérios fornecidos pelos argumentos. No iOS, os dois últimos argumentos são sempre os mesmos. (Essa função é emprestada do OS X, em que há muito mais opções.) O primeiro argumento é uma constante que especifica o diretório na área restrita para o qual você deseja o caminho. Por exemplo, buscar por **NSCachesDirectory** retornará o diretório Caches na área restrita do aplicativo.

Você pode consultar a documentação de uma das constantes – como `NSDocumentDirectory` – para localizar as outras opções. Lembre-se de que essas constantes são compartilhadas pelo iOS e pelo OS X, então, nem todas funcionarão no iOS.

O valor de retorno de `NSSearchPathForDirectoriesInDomains` é um array de strings. Trata-se de um array de strings porque, no OS X, podem existir diversos caminhos que atendem aos critérios de busca. Entretanto, no iOS, haverá apenas um (se o diretório pelo qual você buscou for um diretório de área restrita apropriado). Portanto, o nome do arquivo de arquivamento é anexado ao primeiro e único caminho no array. Esse será o local onde o arquivo das instâncias de `BNRItem` residirão.

## NSKeyedArchiver e NSKeyedUnarchiver

Agora, você tem um local para salvar os dados no sistema de arquivos e um objeto de modelo que pode ser salvo no sistema de arquivos. As duas perguntas finais são: como você inicia os processos de salvamento e carregamento, e quando faz isso? Para salvar as instâncias de `BNRItem`, você usará a classe `NSKeyedArchiver` quando o aplicativo for “fechado”.

No `BNRItemStore.h`, declare um novo método.

```
- (BOOL)saveChanges;
```

Implemente esse método no `BNRItemStore.m` para enviar a mensagem `archiveRootObject:toFile:` para a classe `NSKeyedArchiver`.

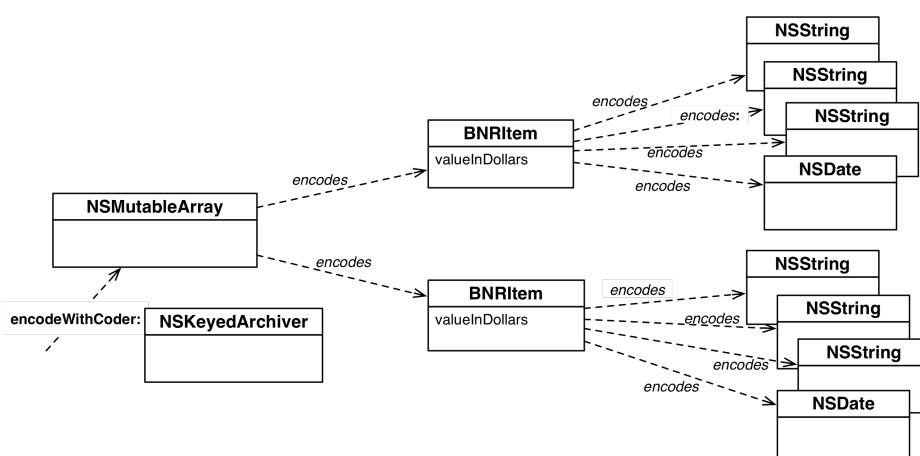
```
- (BOOL)saveChanges
{
    NSString *path = [self itemArchivePath];

    // Returns YES on success
    return [NSKeyedArchiver archiveRootObject:self.privateItems
                                         toFile:path];
}
```

O método `archiveRootObject:toFile:` tem o cuidado de salvar cada `BNRItem` em `privateItems` no `itemArchivePath`. Sim, isso é simples. Veja como o `archiveRootObject:toFile:` funciona:

- O método começa criando uma instância de `NSKeyedArchiver`. (`NSKeyedArchiver` é uma subclasse concreta da classe abstrata  `NSCoder`.)
- `privateItems` recebe a mensagem `encodeWithCoder:` e a instância de `NSKeyedArchiver` é passada como um argumento.
- O array `privateItems` envia então `encodeWithCoder:` a todos os objetos que ele contém, passando a mesma `NSKeyedArchiver`. Portanto, todas as instâncias de `BNRItem` codificam suas variáveis de instância exatamente na mesma `NSKeyedArchiver` (Figure 18.4).
- A `NSKeyedArchiver` grava em `path` os dados que coletou.

Figure 18.4 Arquivamento do array `privateItems`



Quando o usuário pressiona o botão Home no dispositivo, a mensagem `applicationDidEnterBackground:` é enviada para `BNRAppDelegate`. Isso ocorre quando desejamos enviar `saveChanges` para `BNRItemStore`.

No `BNRAppDelegate.m`, implemente `applicationDidEnterBackground:` para iniciar o salvamento dos itens. Certifique-se de importar o arquivo de cabeçalho da `BNRItemStore` no topo desse arquivo.

```
#import "BNRItemStore.h"

@implementation BNRAppDelegate

- (void)applicationDidEnterBackground:(UIApplication *)application
{
    BOOL success = [[BNRItemStore sharedStore] saveChanges];
    if (success) {
        NSLog(@"Saved all of the BNRItems");
    }
    else {
        NSLog(@"Could not save any of the BNRItems");
    }
}
```

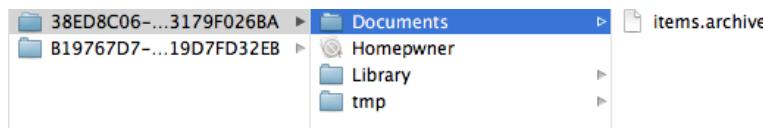
(Talvez esse método já tenha sido implementado pelo template. Em caso afirmativo, adicione código ao método existente em vez de escrever um totalmente novo.)

Compile e execute o aplicativo no simulador. Crie algumas instâncias de `BNRItem`. Em seguida, pressione o botão Home para sair do aplicativo. Verifique o console. Você verá uma instrução de registro indicando que os itens foram salvos.

Embora você ainda não possa carregar essas instâncias de `BNRItem` novamente no aplicativo, ainda pode verificar se *alguma coisa* foi salva. No Finder, pressione Command-Shift-G. Depois, digite `~/Library/Application Support/iPhone Simulator` e pressione Enter. Este é o local em que todos os aplicativos e suas áreas restritas são armazenados para o simulador.

Abra o diretório 7.0 (ou, se você estiver trabalhando com outra versão do iOS, selecione esse diretório). Abra Applications para ver a lista de todos os aplicativos que foram executados no simulador usando o iOS 7.0. Infelizmente, esses aplicativos têm nomes que realmente não ajudam muito. Você tem de explorar cada diretório para encontrar aquele que contém o Homepwner.

Figure 18.5 Área restrita do Homepwner



No diretório do Homepwner, navegue até o diretório `Documents` (Figure 18.5). Você verá o arquivo `items.archive`. Uma dica: crie um alias para o diretório do iPhone Simulator em algum lugar conveniente a fim de facilitar a verificação das áreas restritas de seus aplicativos.

Agora, falaremos sobre o carregamento desses arquivos. Para carregar instâncias de `BNRItem` quando o aplicativo for iniciado, você usará a classe `NSKeyedUnarchiver` assim que a `BNRItemStore` for criada.

No `BNRItemStore.m`, adicione o seguinte código ao método `initPrivate`.

```
- (instancetype)initPrivate
{
    self = [super init];
    if (self) {
        _privateItems = [[NSMutableArray alloc] init];
        NSString *path = [self itemArchivePath];
        _privateItems = [NSKeyedUnarchiver unarchiveObjectWithFile:path];
        // If the array hadn't been saved previously, create a new empty one
        if (!_privateItems) {
            _privateItems = [[NSMutableArray alloc] init];
        }
    }
    return self;
}
```

O método `unarchiveObjectWithFile:` criará uma instância de `NSKeyedUnarchiver` e carregará o arquivamento localizado no `itemArchivePath` para essa instância. Desse modo, a `NSKeyedUnarchiver` examinará o tipo do objeto raiz no arquivamento e criará uma instância desse tipo. Nesse caso, o tipo será uma `NSMutableArray`, pois você criou esse arquivamento com um objeto raiz desse tipo. (Se, em vez disso, o objeto raiz fosse uma `BNRItem`, o `unarchiveObjectWithFile:` retornaria uma instância de `BNRItem`.)

A `NSMutableArray` recém-alocada então recebe `initWithCoder:` e, como você pode imaginar, a `NSKeyedUnarchiver` é passada como o argumento. O array começa a decodificar seu conteúdo (instâncias de `BNRItem`) com base em `NSKeyedUnarchiver` e envia a cada um desses objetos a mensagem `initWithCoder:` passando a mesma `NSKeyedUnarchiver`.

Agora você pode compilar e executar o aplicativo. Quaisquer itens que um usuário informar estarão disponíveis até que ele os exclua explicitamente. Uma observação sobre o teste de seu código para salvamento e carregamento: Se você desativar Homepwner no Xcode, o `applicationDidEnterBackground:` não poderá ser chamado, e o array de itens não será salvo. Primeiro, você deve pressionar o botão Home e desativá-lo no Xcode, clicando no botão Stop.

Agora que você pode salvar e carregar itens, não há motivo para preencher automaticamente cada um deles com dados aleatórios. No `BNRItemStore.m`, modifique a implementação de `createItem` para que ele crie uma `BNRItem` vazia, em vez de uma com dados aleatórios.

```
- (BNRItem *)createItem
{
    BNRItem *item = [BNRItem randomItem];
    BNRItem *item = [[BNRItem alloc] init];

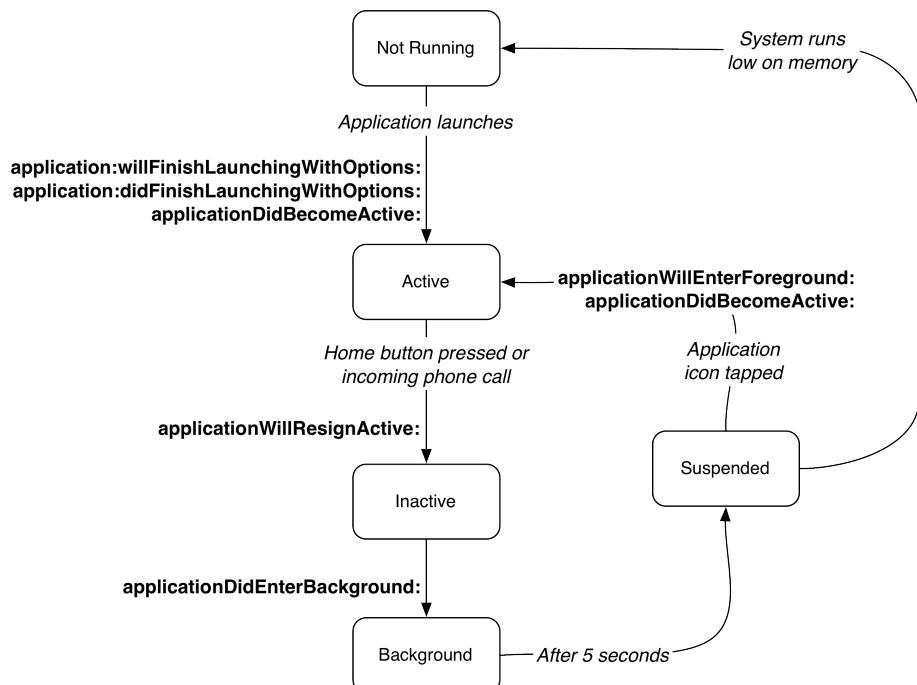
    [self.privateItems addObject:item];

    return item;
}
```

## Transições e estados do aplicativo

No Homepwner, os itens são arquivados quando o aplicativo entra no *estado de segundo plano*. É importante entender todos os estados em que um aplicativo pode estar, o que faz com que ele faça a transição entre estados e como seu código pode ser notificado sobre essas transições. Essas informações são resumidas na Figure 18.6.

Figure 18.6 Estados de um aplicativo típico



Quando um aplicativo não está em execução, ele está no estado *não rodando* e não executa nenhum código ou não tem nenhuma memória reservada na RAM.

Depois que o usuário inicia um aplicativo, ele entra no *estado ativo*. Quando está no estado ativo, a interface do aplicativo aparece na tela e aceita eventos, e seu código lida com esses eventos.

Enquanto estiver no estado ativo, um aplicativo poderá ser temporariamente interrompido por um evento de sistema, como uma mensagem SMS, uma notificação push, uma chamada telefônica ou um alarme. Uma sobreposição aparecerá no topo de seu aplicativo para lidar com esse evento. Esse estado é conhecido como *estado inativo*. No estado inativo, um aplicativo é visível em grande parte (uma visão de alerta aparecerá e ocultará parte da interface) e executa códigos, mas não recebe eventos. Geralmente, os aplicativos passam pouco tempo no estado inativo. Você pode colocar um aplicativo ativo no estado inativo pressionando o botão Lock no topo do dispositivo. O aplicativo ficará inativo até que o dispositivo seja desbloqueado.

Quando o usuário pressiona o botão Home ou alterna para outro aplicativo de alguma outra forma, o aplicativo entra no *estado de segundo plano*. (Na verdade, ele fica poucos instantes no estado inativo antes de fazer a transição para segundo plano.) Em segundo plano, a interface de um aplicativo não é visível nem recebe eventos, mas ainda pode executar códigos. Por padrão, um aplicativo que entra no estado de segundo plano tem cerca de dez segundos antes de entrar no *estado suspenso*. Seu aplicativo não deve confiar nesse número; em vez disso, ele deve salvar os dados do usuário e liberar quaisquer recursos compartilhados o mais rápido possível.

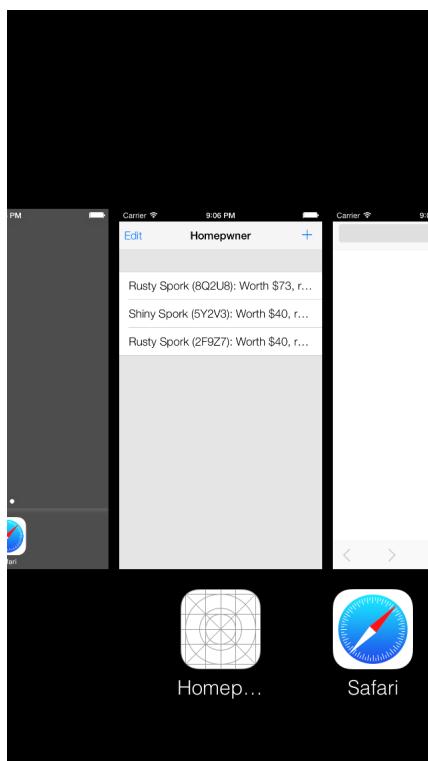
Um aplicativo no estado suspenso não pode executar códigos; você não consegue ver sua interface, e quaisquer recursos de que ele não precisa enquanto está suspenso são destruídos. Um aplicativo suspenso fica basicamente congelado e pode ser rapidamente descongelado quando o usuário iniciá-lo novamente. A Table 18.1 resume as características dos diferentes estados do aplicativo.

Table 18.1 Estados do aplicativo

| Estado        | Visível         | Recebe eventos | Executa códigos |
|---------------|-----------------|----------------|-----------------|
| Não rodando   | Não             | Não            | Não             |
| Ativo         | Sim             | Sim            | Sim             |
| Inativo       | Em grande parte | Não            | Sim             |
| Segundo plano | Não             | Não            | Sim             |
| Suspenso      | Não             | Não            | Não             |

Você pode ver quais aplicativos estão em segundo plano ou suspensos clicando duas vezes no botão Home para acessar a tela de multitarefas. (Aplicativos executados recentemente e que foram encerrados também podem aparecer nessa tela.)

Figure 18.7 Aplicativos em segundo plano e suspensos da tela de multitarefas



Um aplicativo no estado suspenso permanecerá nesse estado enquanto houver memória adequada no sistema. Quando o sistema operacional decidir que a memória está ficando baixa, ele encerrará os aplicativos suspensos conforme a necessidade. Um aplicativo suspenso não obtém nenhuma notificação de que será encerrado; ele simplesmente é removido da memória. (Um aplicativo pode permanecer na tela de multitarefas depois de ter sido encerrado, mas deverá ser reiniciado ao ser tocado.)

Quando um aplicativo muda seu estado, o delegate do aplicativo recebe uma mensagem. Veja algumas das mensagens do protocolo `UIApplicationDelegate` que anunciam as transições de estado do aplicativo. (Elas são exibidas na Figure 18.6.)

```
- (BOOL)application:(UIApplication *)app
didFinishLaunchingWithOptions:(NSDictionary *)options

- (void)applicationDidBecomeActive:(UIApplication *)app;
- (void)applicationWillResignActive:(UIApplication *)app;
- (void)applicationDidEnterBackground:(UIApplication *)app;
- (void)applicationWillEnterForeground:(UIApplication *)app;
```

Você pode implementar códigos nesses métodos para realizar as ações apropriadas para seu aplicativo. Fazer a transição para o estado de segundo plano é um bom lugar para salvar quaisquer alterações pendentes e o estado do aplicativo, pois será a última vez em que seu aplicativo poderá executar códigos antes de entrar no estado suspenso. No estado suspenso, um aplicativo poderá ser encerrado quando o sistema operacional julgar necessário.

## Gravação no sistema de arquivos com NSData

Seu arquivamento no Homepwner salva e carrega a `itemKey` em cada `BNRItem`, mas e as imagens? Vamos estender o armazenamento de imagens para salvar as imagens conforme elas forem adicionadas e buscá-las quando forem necessárias.

As imagens das instâncias de `BNRItem` também deverão ser armazenadas no diretório `Documents`. Você pode usar a chave de imagem gerada quando o usuário utiliza uma figura para nomear a imagem no sistema de arquivos.

Abra o `BNRImageStore.m` e adicione uma nova declaração de método à extensão da classe.

```
- (NSString *)imagePathForKey:(NSString *)key;
```

Implemente `imagePathForKey`: no `BNRImageStore.m` para criar um caminho no diretório Documents usando uma chave específica.

```
- (NSString *)imagePathForKey:(NSString *)key
{
    NSArray *documentDirectories =
        NSSearchPathForDirectoriesInDomains(NSDocumentDirectory,
                                             NSUserDomainMask,
                                             YES);

    NSString *documentDirectory = [documentDirectories firstObject];
    return [documentDirectory stringByAppendingPathComponent:key];
}
```

Para salvar e carregar uma imagem, você copiará a representação JPEG da imagem em um buffer na memória. Em vez de simplesmente alocar um buffer, os programadores de Objective-C têm uma classe útil para criar, manter e destruir esses tipos de buffer: `NSData`. Uma instância de `NSData` contém alguns bytes de dados binários, e você usará `NSData` para armazenar dados de imagem.

No `BNRImageStore.m`, modifique `setImage:ForKey:` para obter um caminho e salvar a imagem.

```
- (void)setImage:(UIImage *)image forKey:(NSString *)key
{
    self.dictionary[key] = image;

    // Create full path for image
    NSString *imagePath = [self imagePathForKey:key];

    // Turn image into JPEG data
    NSData *data = UIImageJPEGRepresentation(image, 0.5);

    // Write it to full path
    [data writeToFile:imagePath atomically:YES];
}
```

Vamos examinar esse código mais detalhadamente. A função `UIImageJPEGRepresentation` utiliza dois parâmetros: uma `UIImage` e uma qualidade de compressão. A qualidade de compressão é um float de 0 a 1, em que 1 é a qualidade mais alta (compressão mínima). A função retorna uma instância de `NSData`.

Essa instância de `NSData` pode ser gravada no sistema de arquivos enviando a ele a mensagem `writeToFile:atomically:`. Os bytes contidos nessa `NSData` são então gravados no caminho especificado pelo primeiro parâmetro. O segundo parâmetro, `atomically`, é um valor booleano. Se for YES, o arquivo será gravado em um local temporário no sistema de arquivos e, quando a operação de gravação for concluída, esse arquivo será renomeado com o caminho do primeiro parâmetro, substituindo qualquer arquivo existente anteriormente. A gravação atômica impede a corrupção de dados caso seu aplicativo falhe durante o processo de gravação.

Vale notar que esse modo de gravar dados no sistema de arquivos *não* é um arquivamento. Embora instâncias de `NSData` possam ser arquivadas, usar o método `writeToFile:atomically:` copia os bytes da `NSData` diretamente no sistema de arquivos.

No `BNRImageStore.m`, certifique-se de que, quando uma imagem for excluída do armazenamento, ela também seja excluída do sistema de arquivos:

```
- (void)deleteImageForKey:(NSString *)key
{
    if (!key) {
        return;
    }
    [self.dictionary removeObjectForKey:key];

    NSString *imagePath = [self imagePathForKey:key];
    [[NSFileManager defaultManager] removeItemAtPath:imagePath
                                              error:nil];
}
```

Observe que a imagem é armazenada no sistema de arquivos, e a **BNRImageStore** precisará carregar essa imagem quando for solicitado. O método de classe **imageWithContentsOfFile:** de **UIImage** lerá uma imagem a partir de um arquivo, mediante um caminho.

No **BNRImageStore.m**, substitua o método **imageForKey:** de modo que **BNRImageStore** carregue a imagem a partir do sistema de arquivos caso ainda não tenha feito isso.

```
- (UIImage *)imageForKey:(NSString *)key
{
    return self.dictionary[key];

    // If possible, get it from the dictionary
    UIImage *result = self.dictionary[key];

    if (!result) {
        NSString *imagePath = [self imagePathForKey:key];

        // Create UIImage object from file
        result = [UIImage imageWithContentsOfFile:imagePath];

        // If we found an image on the file system, place it into the cache
        if (result) {
            self.dictionary[key] = result;
        }
        else {
            NSLog(@"Error: unable to find %@", [self imagePathForKey:key]);
        }
    }
    return result;
}
```

Compile e execute o aplicativo novamente. Tire uma foto de um item, saia do aplicativo e, depois, pressione o botão Home. Inicie o aplicativo novamente. Ao selecionar esse mesmo item, serão apresentados seus detalhes salvos – incluindo a foto que você acabou de tirar.

Observe também que as imagens foram salvas logo depois que foram tiradas, e as instâncias de **BNRItem** foram salvas somente quando o aplicativo entrou em segundo plano. Salve as imagens logo depois porque elas são muito grandes para serem mantidas na memória por muito tempo.

## NSNotificationCenter e avisos de pouca memória

Quando o sistema está ficando com pouca RAM, ele emite um aviso de pouca memória para o aplicativo em execução. O aplicativo responde liberando quaisquer recursos que não sejam necessários no momento e que possam ser facilmente recriados. Os controladores de visão, durante um aviso de pouca memória, recebem a mensagem **didReceiveMemoryWarning**.

Objetos que não sejam controladores de visão podem ter dados que não estejam usando e recriá-los posteriormente. A **BNRImageStore** é um desses objetos – quando ocorre um aviso de pouca memória, ela consegue liberar sua propriedade das imagens esvaziando seu **dictionary**. Então, se outro objeto solicitar uma imagem específica novamente, essa imagem poderá ser carregada na memória com base no sistema de arquivos.

Para fazer objetos que não são controladores de visão responderem a avisos de pouca memória, deve-se usar a central de notificações. Todos os aplicativos têm uma instância de **NSNotificationCenter**, que trabalha como um mural inteligente. Um objeto pode se registrar como observador (“Envie-me notificações sobre ‘cães perdidos’”). Quando outro objeto posta uma notificação (“Perdi meu cão”), a central de notificações encaminha a notificação aos observadores registrados.

Sempre que ocorrer um aviso de pouca memória, **UIApplicationDidReceiveMemoryWarningNotification** será enviada à central de notificações. Os objetos que desejam implementar seus próprios manipuladores de aviso de pouca memória podem se registrar para essa notificação.

No **BNRImageStore.m**, edite o método **init** para registrar o armazenamento de imagens como um observador dessa notificação.

```

- (instancetype)initPrivate
{
    self = [super init];
    if (self) {
        _dictionary = [[NSMutableDictionary alloc] init];

        NSNotificationCenter *nc = [NSNotificationCenter defaultCenter];
        [nc addObserver:self
            selector:@selector(clearCache:)
            name:UIApplicationDidReceiveMemoryWarningNotification
            object:nil];

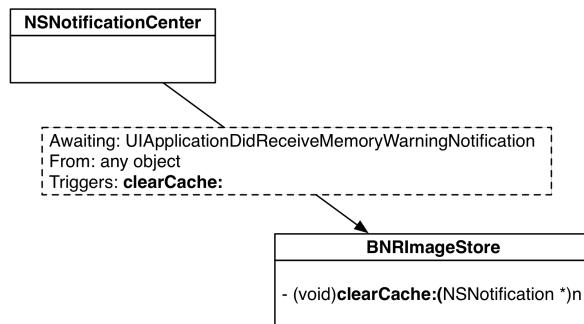
    }

    return self;
}

```

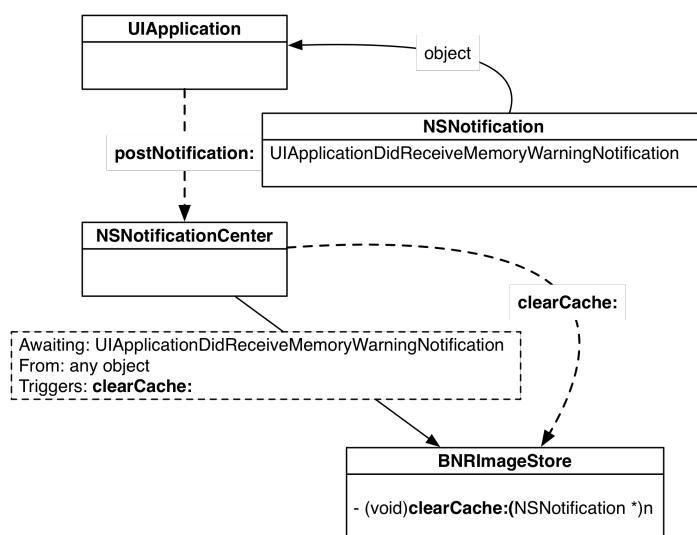
Agora, seu armazenamento de imagens está registrado como observador na central de notificações (Figure 18.8).

Figure 18.8 Registrado como observador na central de notificações



Agora, quando um aviso de pouca memória for publicado, a central de notificações enviará a mensagem **clearCache:** para a instância de **BNRImageStore** (Figure 18.9).

Figure 18.9 Re却imento de notifica莽ao



No **BNRImageStore.m**, implemente **clearCache:** para remover todas as instâncias de **UIImage** do **dictionary** de **BNRImageStore**.

```

- (void)clearCache:(NSNotification *)note
{
    NSLog(@"flushing %d images out of the cache", [self.dictionary count]);
    [self.dictionary removeAllObjects];
}

```

Remover um objeto de um dicionário abdicará a propriedade do objeto, portanto, limpar o cache fará com que todas as imagens percam um proprietário. As imagens que não estiverem sendo usadas por outros objetos serão destruídas e, quando forem solicitadas novamente, elas serão recarregadas do sistema de arquivos. Se uma imagem estiver atualmente sendo exibida na imageView de **BNRDetailViewController**, ela não será destruída, pois é de propriedade da imageView. Quando a imageView da **BNRDetailViewController** perder a propriedade dessa imagem (porque **BNRDetailViewController** desapareceu da pilha ou porque uma nova imagem foi capturada), ela será destruída. Ela será recarregada posteriormente, se necessário.

Compile e execute o aplicativo no simulador. Crie ou carregue algumas imagens. Em seguida, selecione Simulate Memory Warning no menu Hardware. Você verá uma instrução de registro indicando que o cache foi esvaziado.

## Mais informações sobre NSNotificationCenter

As notificações são outra forma de *callbacks*, como delegação e pares destino-ação. Contudo, diferentemente da delegação e dos pares destino-ação, que requerem que o objeto responsável pelo evento envie uma mensagem diretamente para seu delegate ou destinos, as notificações usam um intermediário: a **NSNotificationCenter**.

As notificações do Objective-C são representadas pelas instâncias de **NSNotification**. Cada **NSNotification** possui um nome (usado pela central de notificações para localizar observadores), um objeto (responsável por publicar a notificação) e um dicionário opcional com informações de usuários, que contém dados adicionais dos quais o publicador deseja que os observadores tenham conhecimento. Por exemplo, se por algum motivo o frame da barra de status for alterado, a **UIApplication** publicará uma **UIApplicationDidChangeStatusBarFrameNotification** com um dicionário de informações de usuários. No dicionário, está o novo frame da barra de status. Se você tiver recebido a notificação, pode obter o frame desta maneira:

```

- (void)statusBarMovedOrResized:(NSNotification *)note
{
    NSDictionary *userInfo = [note userInfo];
    NSValue *wrappedRect = userInfo[UIApplicationStatusBarFrameUserInfoKey];
    CGRect newFrame = [wrappedRect CGRectValue];

    ...use frame here...
}

```

Observe que o **CGRect** precisou ser incluído em uma **NSValue** porque apenas objetos podem entrar em dicionários.

Como é possível saber o que há no dicionário **userInfo**? Cada notificação é documentada na referência de classe. A maioria afirma que: “esta notificação não contém um dicionário **userInfo**”. No caso de notificações com dicionários **userInfo**, todas as chaves e o que é mapeado serão listados.

O último argumento de **addObserver:selector:name:object:** geralmente é **nil** – que significa que, independentemente do objeto que publicou uma notificação “fogo!”, o observador conseguirá enviar sua mensagem. É possível designar um ponteiro a um objeto para esse argumento, e o observador será notificado apenas se esse objeto publicar a notificação na qual está registrado, e qualquer outro objeto que publicar a mesma notificação será ignorado.

Um dos objetivos da central de notificações é permitir que múltiplos objetos registrem um callback para o mesmo evento. Qualquer quantidade de objetos pode se registrar como observador para o mesmo nome de notificação. Quando a notificação ocorrer, todos esses objetos receberão a mensagem na qual estão registrados (sem ordem específica). Portanto, as notificações são uma boa solução quando mais de um objeto está interessado em um evento. Por exemplo, muitos objetos podem querer saber sobre um evento de rotação, então a Apple usa notificações para isso.

E por último: a **NSNotificationCenter** não tem nada a ver com comunicação interaplicativos, notificações push ou notificações locais. Trata-se simplesmente de comunicação entre objetos em um único aplicativo.

## Padrão de projeto Modelo-Visão-Controlador-Armazenamento

Neste exercício, você expandiu a **BNRItemStore** para permitir que ela salve e carregue instâncias de **BNRItem** a partir do sistema de arquivos. O objeto de controlador solicita a **BNRItemStore** aos objetos de modelo de que precisa, mas não tem de se preocupar com o local de onde esses objetos vieram. Em se tratando do controlador, quando ele quer um objeto, ele o obtém; a **BNRItemStore** é responsável por garantir que isso aconteça.

O padrão de projeto MVC (Modelo-Visão-Controlador) exige que o controlador tenha a responsabilidade de salvar e carregar objetos de modelo. No entanto, na prática, isso pode ser devastador; o controlador simplesmente fica muito ocupado com a manipulação das interações entre objetos de modelo e de visão para lidar com os detalhes de como os objetos são obtidos e salvos. Desse modo, é útil mover para outro tipo de objeto a lógica que lida com o local de onde os objetos de modelo vêm e com o local onde eles são salvos: um *armazenamento*.

Um armazenamento apresenta uma série de métodos que permite que um objeto de controlador obtenha e salve objetos de modelo. Os detalhes de onde esses objetos de modelo vêm ou de como eles são obtidos são de responsabilidade do armazenamento. Neste capítulo, o armazenamento trabalhou com um arquivo simples. No entanto, o armazenamento também poderia acessar um banco de dados, comunicar-se com um serviço web ou usar outro método para produzir os objetos de modelo para o controlador.

Um benefício dessa abordagem, além das classes de controlador simplificadas, é que você pode alterar *como* o armazenamento funciona sem modificar o controlador ou o restante de seu aplicativo. Essa modificação pode ser simples, como a estrutura de diretórios dos dados, ou pode ser muito maior, como o formato dos dados. Desse modo, se um aplicativo tiver mais de um objeto de controlador que precise salvar e carregar dados, você terá apenas de alterar o objeto de armazenamento.

Muitos desenvolvedores falam a respeito do padrão de projeto Modelo-Visão-Controlador. Neste capítulo, estendemos a ideia para um padrão de projeto *Modelo-Visão-Controlador-Armazenamento*.

## Desafio de bronze: PNG

Em vez de salvar cada imagem como JPEG, salve-as como PNG.

## Para os mais curiosos: transições de estado do aplicativo

Vamos escrever um código rápido para entender melhor as diferentes transições de estado do aplicativo.

Você já conhece a `self`, uma variável implícita que aponta para a instância que está executando o método atual. Há outra variável implícita chamada `_cmd`, que é o seletor do método atual. Você pode obter a representação da `NSString` de um seletor usando a função `NSStringFromSelector`.

No `BNRAppDelegate.m`, implemente os métodos delegate de transição de estado do aplicativo para que eles exibam o nome do método. Você precisará adicionar mais quatro métodos. (Verifique se o template já não criou esses métodos antes de escrever os novos.)

```
- (void)applicationWillResignActive:(UIApplication *)application
{
    NSLog(@"%@", NSStringFromSelector(_cmd));
}

- (void)applicationDidEnterForeground:(UIApplication *)application
{
    NSLog(@"%@", NSStringFromSelector(_cmd));
}

- (void)applicationDidBecomeActive:(UIApplication *)application
{
    NSLog(@"%@", NSStringFromSelector(_cmd));
}

- (void)applicationWillTerminate:(UIApplication *)application
{
```

```

    NSLog(@"%@", NSStringFromSelector(_cmd));
}

```

Agora, adicione as seguintes instruções **NSLog** na parte superior de **application:didFinishLaunchingWithOptions:** e **applicationDidEnterBackground:**.

```

- (BOOL)application:(UIApplication *)application
    didFinishLaunchingWithOptions:(NSDictionary *)launchOptions
{
    NSLog(@"%@", NSStringFromSelector(_cmd));
    ...
}

- (void)applicationDidEnterBackground:(UIApplication *)application
{
    NSLog(@"%@", NSStringFromSelector(_cmd));
    [[BNRItemStore sharedStore] saveChanges];
}

```

Compile e execute o aplicativo. Você verá que o aplicativo recebe as mensagens **application:didFinishLaunchingWithOptions:** e **applicationDidBecomeActive:**. Faça alguns testes para ver quais ações causam quais transições.

Pressione o botão Home. O console informará que o aplicativo ficou rapidamente inativo e depois entrou no estado de segundo plano. Reinicie o aplicativo tocando em seu ícone na tela Home ou na tela de multitarefas. O console informará que o aplicativo entrou em primeiro plano e ficou ativo.

Pressione o botão Home para sair novamente do aplicativo. Em seguida, clique duas vezes no botão Home para abrir a tela de multitarefas. Arraste o aplicativo Homepwner para cima e para fora da tela, para sair do aplicativo. Observe que nenhuma mensagem é enviada ao delegate de seu aplicativo neste momento – ele simplesmente encerrou.

## Para os mais curiosos: leitura e gravação no sistema de arquivos

Além dos métodos de arquivamento e de leitura e gravação de binários da **NSData**, existem alguns outros métodos para transferir dados para o sistema de arquivos e a partir dele. Um deles, o Core Data, será tratado no Chapter 23. Vale a pena falar de alguns outros aqui.

Você tem acesso às funções padrão de E/S (I/O) de arquivo na biblioteca do C. As funções serão semelhantes ao seguinte:

```

FILE *inFile = fopen("textfile", "rt");
char *buffer = malloc(someSize);
fread(buffer, byteCount, 1, inFile);

FILE *outFile = fopen("binaryfile", "w");
fwrite(buffer, byteCount, 1, outFile);

```

No entanto, você não verá essas funções sendo muito usadas, pois há maneiras mais convenientes de ler e gravar dados binários e de texto. Usar a **NSData** funciona bem para dados binários. Para dados de texto, a **NSString** tem dois métodos de instância: **writeToFile:atomically:encoding:error:** e **initWithContentsOfFile:**. Eles são usados desta maneira:

```

// A local variable to store an error object if one comes back
NSError *err;

NSString *someString = @"Text Data";
BOOL success = [someString writeToFile:@"/some/path/file"
                           atomically:YES
                           encoding:NSUTF8StringEncoding
                           error:&err];

if (!success) {
    NSLog(@"Error writing file: %@", [err localizedDescription]);
}

```

```
NSString *myEssay = [[NSString alloc] initWithContentsOfFile:@"/some/path/file"
                                                encoding:NSUTF8StringEncoding
                                                error:&err];
if (!myEssay) {
    NSLog(@"Error reading file: %@", [err localizedDescription]);
}
```

O que é esse objeto de **NSError**? Alguns métodos podem falhar por uma série de motivos. Por exemplo, a gravação no sistema de arquivos pode falhar porque o caminho é inválido ou porque o usuário não tem permissão para gravar no caminho especificado. Um objeto de **NSError** contém o motivo de uma falha. Você pode enviar a mensagem **localizedDescription** a uma instância de **NSError** para obter uma descrição do erro legível por humanos. Isso é algo que você pode mostrar ao usuário ou exibir em um console de depuração.

A sintaxe para obter de volta uma instância de **NSError** é um pouco estranha. Um objeto de erro será criado apenas se tiver ocorrido um erro; caso contrário, o objeto não será necessário. Quando um método pode retornar um erro por meio de um de seus argumentos, você cria uma variável local que seja um ponteiro para um objeto de **NSError**. Observe que você não criou instâncias do objeto de erro – esse é o trabalho do método que você está chamando. Em vez disso, você passa o *endereço* de sua variável de ponteiro (**&err**) para o método que pode gerar um erro. Se um erro ocorrer na implementação desse método, uma instância de **NSError** será criada, e seu ponteiro será configurado para apontar para esse novo objeto. Se você não precisar do objeto de erro, sempre poderá passar **nil**.

Muitas vezes, você deseja mostrar o erro ao usuário. Isso geralmente é feito com uma **UIAlertView** (Figure 18.10).

Figure 18.10 Exemplo de **UIAlertView**



A criação de **UIAlertView** é feita desta forma:

```
NSString *x = [[NSString alloc] initWithContentsOfFile:@"/some/path/file"
                                                encoding:NSUTF8StringEncoding
                                                error:&err];
if (!x) {
    UIAlertView *a = [[UIAlertView alloc] initWithTitle:@"Read Failed"
                                                message:[err localizedDescription]
                                                delegate:nil
                                                cancelButtonTitle:@"OK"
                                                otherButtonTitles:nil];
    [a show];
}
```

Observe que, em muitas linguagens, qualquer coisa inesperada resulta no disparo de uma exceção. Entre programadores de Objective-C, exceções são quase sempre usadas para indicar erro do programador. Quando uma exceção é disparada, as informações sobre o que deu errado ficam em um objeto de **NSEException**. Essas informações geralmente são uma dica para o programador, por exemplo: “você tentou acessar o sétimo objeto neste array, mas há apenas dois”. Os símbolos da pilha de chamadas (conforme aparecem quando há disparo de exceção) também estão na **NSEException**.

Quando você usa a **NSEException** e quando você usa a **NSError**? Se você estiver escrevendo um método que deverá ser chamado apenas com um número ímpar como argumento, dispare uma exceção caso seja chamado com um número par – o chamador está cometendo um erro e você deseja ajudar o programador a reconhecer seu erro. Se você estiver escrevendo um método que deseja ler o conteúdo de um diretório em particular, mas não possui os privilégios necessários, crie uma **NSError** e passe-o de volta para o chamador para indicar o motivo pelo qual você não conseguiu atender a essa solicitação bastante razoável.

Assim como **NSString**, as classes **NSDictionary** e **NSArray** têm métodos **writeToFile:** e **initWithContentsOfFile:**. Para gravar objetos de coleção no sistema de arquivos com esses métodos, esses objetos devem conter apenas objetos *serializáveis com lista de propriedades*. Os únicos objetos que são serializáveis com lista de propriedades são: **NSString**, **NSNumber**, **NSDate**, **NSData**, **NSArray** e **NSDictionary**. Quando uma **NSArray** ou **NSDictionary** é gravada no sistema de arquivos com esses métodos, uma *lista de propriedades XML* é criada. Uma lista de propriedades XML é uma coleção de valores marcados:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN"
    "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<array>
    <dict>
        <key>firstName</key>
        <string>Christian</string>
        <key>lastName</key>
        <string>Keur</string>
    </dict>
    <dict>
        <key>firstName</key>
        <string>Joe</string>
        <key>lastName</key>
        <string>Conway</string>
    </dict>
    <dict>
        <key>firstName</key>
        <string>Aaron</string>
        <key>lastName</key>
        <string>Hillegass</string>
    </dict>
</array>
</plist>
```

As listas de propriedades XML são um modo conveniente de armazenar dados, pois podem ser lidas em praticamente qualquer sistema. Muitos aplicativos de serviços web usam listas de propriedades como entrada e saída. O código para gravar e ler uma lista de propriedades será semelhante ao seguinte:

```
NSMutableDictionary *d = [NSMutableDictionary dictionary];
d[@"String"] = @"A string";
[d writeToFile:@"/some/path/file" atomically:YES];

NSMutableDictionary *anotherD = [[NSMutableDictionary alloc]
    initWithContentsOfFile:@"/some/path/file"];
```

## Para os mais curiosos: o pacote de aplicativo

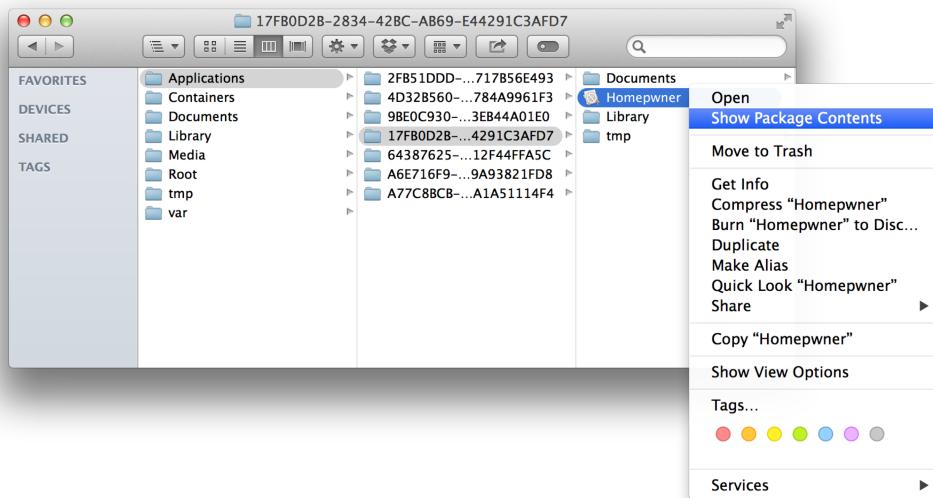
Ao compilar um projeto de aplicativo para iOS no Xcode, você cria um *pacote de aplicativo*. O pacote de aplicativo contém o executável do aplicativo e quaisquer recursos que acompanham seu aplicativo. Recursos são coisas como: arquivos XIB, imagens e arquivos de áudio – quaisquer arquivos usados no tempo de execução. Quando você adicionar um arquivo de recurso a um projeto, o Xcode será inteligente o suficiente para perceber que ele deve ser empacotado com seu aplicativo.

Como você pode informar quais arquivos serão empacotados com seu aplicativo? Selecione o projeto Homepwner no navegador de projetos. Verifique o painel Build Phases no destino Homepwner. Tudo o que estiver em Copy Bundle Resources será adicionado ao pacote de aplicativo durante a compilação.

Cada item no grupo de destino Homepwner é uma das fases que ocorre quando um projeto é compilado. A fase Copy Bundle Resources é quando todos os recursos em seu projeto são copiados para o pacote de aplicativo.

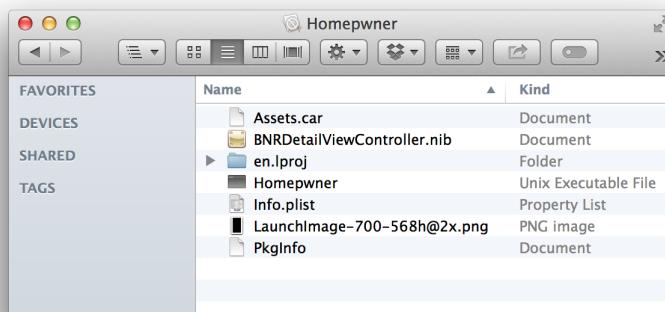
Você pode verificar como ficará um pacote de aplicativo no sistema de arquivos depois de instalar um aplicativo no simulador. Navegue para `~/Library/Application Support/iPhone Simulator/(version number)/Applications`. Os diretórios contidos nesse diretório são as áreas restritas do aplicativo para os aplicativos instalados no simulador do iOS de seu computador. Abrir um desses diretórios exibirá o que você espera em uma área restrita de aplicativo: um pacote de aplicativo e os diretórios `Documents`, `tmp` e `Library`. Clique com o botão direito no pacote de aplicativo (ou clique pressionando a tecla Command) e selecione Show Package Contents no menu contextual (Figure 18.11).

Figure 18.11 Exibição de um pacote de aplicativo



Uma janela do Finder será exibida, mostrando o conteúdo do pacote de aplicativo (Figure 18.12). Quando um usuário fizer download de seu aplicativo da App Store, esses arquivos serão copiados para seu dispositivo.

Figure 18.12 O pacote de aplicativo



É possível carregar arquivos do pacote de aplicativo no tempo de execução. Para obter o caminho completo dos arquivos contidos no pacote de aplicativo, você precisará de um ponteiro para o pacote de aplicativo e depois solicitará a ele o caminho de um recurso.

```
// Get a pointer to the application bundle
NSBundle *applicationBundle = [NSBundle mainBundle];

// Ask for the path to a resource named myImage.png in the bundle
NSString *path = [applicationBundle pathForResource:@"myImage"
                                             ofType:@"png"];
```

Se solicitar o caminho de um arquivo que não esteja no pacote de aplicativo, esse método retornará `nil`. Se o arquivo existir, será retornado o caminho completo, e você poderá usá-lo para carregar o arquivo com a classe apropriada.

Além disso, os arquivos dentro do pacote de aplicativo são somente leitura. Você não pode modificá-los nem pode adicionar arquivos dinamicamente ao pacote de aplicativo no tempo de execução. Os arquivos contidos no pacote de aplicativo geralmente são imagens de botão, efeitos sonoros da interface ou o estado inicial de um banco de dados que acompanha seu aplicativo. Você usará esse método nos próximos capítulos para carregar esses tipos de recursos no tempo de execução.

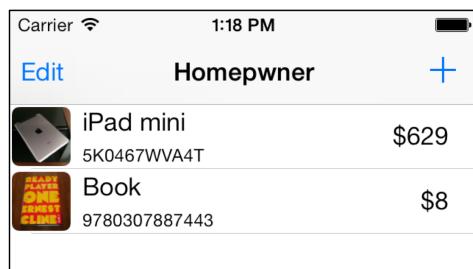
# 19

## Criação de subclasses UITableViewCell

Uma **UITableView** mostra uma lista de objetos **UITableViewCell**. Para muitos aplicativos, a célula básica com seus **textLabel**, **detailTextLabel** e **imageView** é suficiente. Porém, quando você precisa de uma célula com mais detalhes ou um layout diferente, você cria subclasses de **UITableViewCell**.

Neste capítulo, você vai criar uma classe customizada de **UITableViewCell** chamada **BNRItemCell** que exibirá as instâncias **BNRItem** de uma forma mais eficiente. Cada uma dessas células mostrará o nome de uma **BNRItem**, seu valor em dólares, o número de série e uma miniatura de sua imagem (Figure 19.1).

Figure 19.1 Homepwner com células de visão de tabela com subclasses

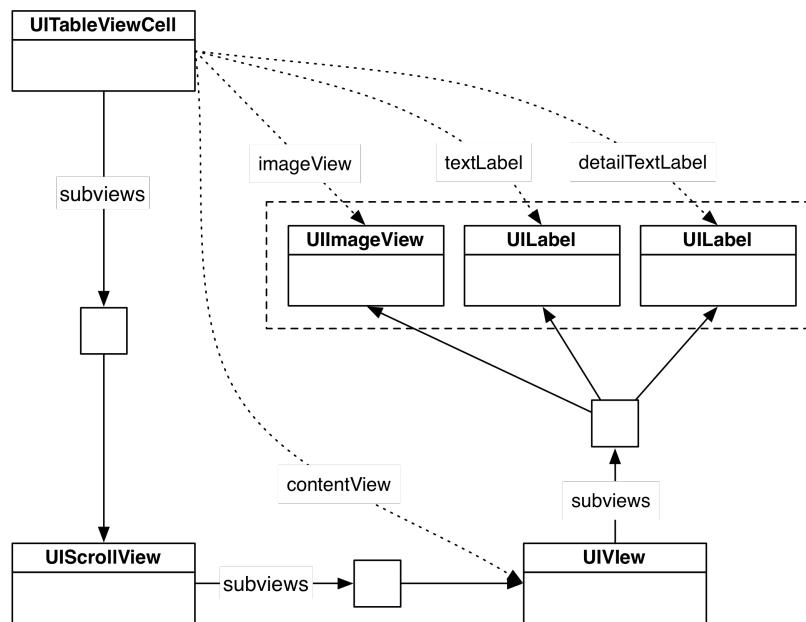


### Criação de BNRItemCell

**UITableViewCell** é uma subclasse **UIView**. Ao criar subclasses **UIView** (ou de qualquer de suas subclasses), você geralmente sobrescreve o respectivo método **drawRect:** para customizar a aparência da visão. Contudo, ao criar subclasses **UITableViewCell**, você geralmente customiza a aparência adicionando subvisões à célula. Contudo, você não as adiciona diretamente à célula; em vez disso, você as adiciona à *visão de conteúdo* da célula.

Cada célula tem uma subvisão chamada **contentView**, que é um recipiente para os objetos de visão que compõem o layout de uma subclasse de célula (Figure 19.2). Quando você cria uma subclasse de **UITableViewCell**, muitas vezes você muda a sua aparência e comportamento alterando as subvisões da **contentView** da célula. Por exemplo, você pode criar instâncias das classes **UITextField**, **UILabel** e **UIButton**, e adicioná-las à **contentView**.

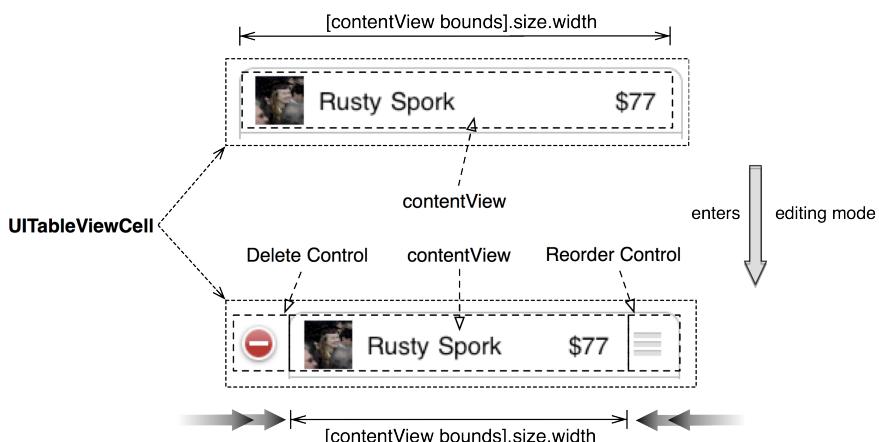
Figure 19.2 Hierarquia UITableViewCell



É importante adicionar subvisões à contentView em vez de diretamente à célula propriamente dita porque a célula redimensionará sua contentView em alguns momentos. Por exemplo, quando uma visão de tabela entra no modo de edição, a contentView se redimensiona para abrir espaço para os controles de edição (Figure 19.3). Se você fosse adicionar subvisões diretamente à **UITableViewCell**, esses controles de edição ocultariam as subvisões. A célula não pode ajustar seu tamanho quando entra no modo de edição (deve se manter com a largura da visão de tabela), mas a contentView pode ser redimensionada, e de fato o é.

(Por falar nisso, você percebeu a **UIScrollView** na hierarquia de células? É assim que o iOS move o conteúdo da célula para a esquerda quando ela entra no modo de edição. Você também pode usar um arraste da direita para a esquerda em uma célula para mostrar o controle de exclusão, e a mesma visão com rolagem é usada para cumprir a tarefa. Faz sentido que a contentView seja uma subvisão da visão com rolagem.)

Figure 19.3 Disposição das células na visão de tabela nos modos standard e de edição



Abra o `Homewner.xcodeproj`. Crie uma nova subclasse **NSObject** e nomeie-a como **BNRItemCell**.

No `BNRItemCell.h`, altere a superclasse para **UITableViewCell**.

```
@interface BNRItemCell : NSObject
@interface BNRItemCell : UITableViewCell
```

## Configuração da interface de uma subclasse UITableViewCell

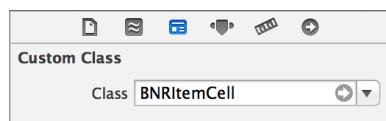
A forma mais fácil de configurar uma subclasse **UITableViewCell** é utilizando um arquivo XIB. Crie um novo arquivo XIB Empty e nomeie-o como `BNRItemCell.xib`. (O Device Family não é relevante para esse arquivo.)

Esse arquivo irá conter uma única instância de **BNRItemCell**. Quando a visão de tabela precisar de uma nova célula, criará uma a partir desse arquivo XIB.

No **BNRItemCell.xib**, selecione **BNRItemCell.xib** e arraste uma instância de **UITableViewCell** da biblioteca de objetos para o canvas. (Certifique-se de selecionar **UITableViewCell**, e não **UITableView** ou **UITableViewcontroller**.)

Selecione Table View Cell na visão de outline e depois selecione o inspetor de identidade (a guia ). Altere o Class para **BNRItemCell** (Figure 19.4).

Figure 19.4 Alteração da classe da célula



Uma **BNRItemCell** mostrará três elementos de texto e uma imagem, então arraste três objetos **UILabel** e um objeto **UIImageView** para a célula. Configure-as conforme mostrado na Figure 19.5. Torne o texto do rótulo inferior um pouco menor e cinza escuro.

Figure 19.5 Layout da **BNRItemCell**



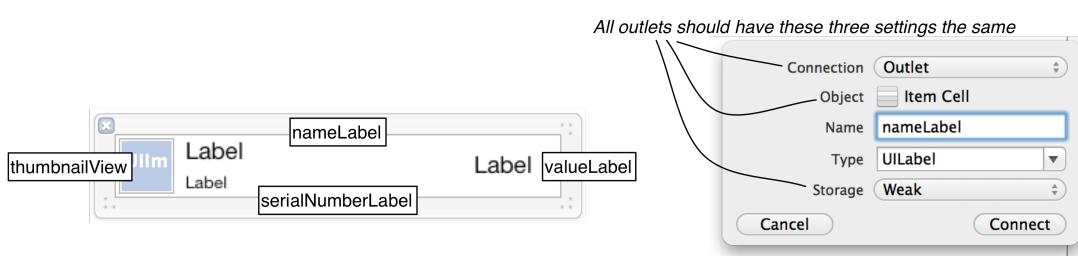
## Exposição das propriedades de **BNRItemCell**

Para que o **BNRItemsViewController** configure o conteúdo de uma **BNRItemCell** no **tableView:cellForRowAtIndexPath:**, a célula deve ter propriedades que exponham os três rótulos e a visão de imagem. Essas propriedades serão definidas por meio de conexões de outlet no **BNRItemCell.xib**.

O próximo passo, portanto, é criar e conectar outlets na **BNRItemCell** para cada uma de suas subvisões. Você usará a mesma técnica que usou nos últimos capítulos para criar os outlets: pressionar Control e arrastar o arquivo XIB para o arquivo-fonte.

Pressione Option e clique em **BNRItemCell.h** com o **BNRItemCell.xib** aberto. Pressione Control e arraste de cada subvisão para a área de declaração de método no **BNRItemCell.h**. Nomeie cada outlet e configure os outros atributos da conexão, conforme mostrado na Figure 19.6. (Preste atenção nos campos Connection, Storage e Object.)

Figure 19.6 Conexões **BNRItemCell**



Verifique se o **BNRItemCell.h** tem a seguinte aparência:

```
@interface BNRItemCell : UITableViewCell

@property (weak, nonatomic) IBOutlet UIImageView *thumbnailView;
@property (weak, nonatomic) IBOutlet UILabel *nameLabel;
@property (weak, nonatomic) IBOutlet UILabel *serialNumberLabel;
@property (weak, nonatomic) IBOutlet UILabel *valueLabel;

@end
```

Observe que você não especificou a classe do File's Owner nem estabeleceu nenhuma conexão a ele. É um pouco diferente de seus arquivos XIB normais, nos quais todas as conexões acontecem entre o File's Owner e os objetos arquivados. Para ver o por quê disso, vamos ver como as células são carregadas no aplicativo.

## Utilização de BNRItemCell

No método `tableView:cellForRowAtIndexPath:` da `BNRItemsViewController`, você criará uma instância de `BNRItemCell` para cada linha da tabela.

No `BNRItemsViewController.m`, importe o arquivo de cabeçalho da `BNRItemCell`, de modo que a `BNRItemsViewController` tenha conhecimento dele.

```
#import "BNRItemCell.h"
```

Anteriormente, você registrou uma classe na visão de tabela para informá-la qual classe deve ser instanciada sempre que ela precisar de uma nova célula de visão de tabela. Agora que você está usando um arquivo NIB customizado para carregar uma subclasse `UITableViewCell`, ao invés disso você registrará o NIB.

No `BNRItemsViewController.m`, modifique o `viewDidLoad` para registrar o `BNRItemCell.xib` para receber um identificador de reutilização "BNRItemCell".

```
- (void)viewDidLoad
{
    [super viewDidLoad];

    [self.tableView registerClass:[UITableViewCell class]
        forCellReuseIdentifier:@"UITableViewCell"];

    // Load the NIB file
    UINib *nib = [UINib nibWithNibName:@"BNRItemCell" bundle:nil];

    // Register this NIB, which contains the cell
    [self.tableView registerNib:nib
        forCellReuseIdentifier:@"BNRItemCell"];
}
```

O registro de um NIB para uma visão de tabela não é nada complicado: a visão de tabela simplesmente armazena a instância de `UINib` em uma `NSDictionary` para a entrada "BNRItemCell". Uma `UINib` contém todos os dados armazenados em seu arquivo XIB e, quando solicitada, pode criar novas instâncias dos objetos contidos nela.

Uma vez que uma `UINib` tenha sido registrada com uma visão de tabela, esta pode ser solicitada a carregar a instância de `BNRItemCell` quando receber o identificador de reutilização "BNRItemCell".

No `BNRItemsViewController.m`, modifique `tableView:cellForRowAtIndexPath:`.

```
- (UITableViewCell *)tableView:(UITableView *)tableView
    cellForRowAtIndexPath:(NSIndexPath *)indexPath
{
    UITableViewCell *cell =
        [tableView dequeueReusableCellWithIdentifier:@"UITableViewCell"
            forIndexPath:indexPath];

    // Get a new or recycled cell
    BNRItemCell *cell =
        [tableView dequeueReusableCellWithIdentifier:@"BNRItemCell"
            forIndexPath:indexPath];

    NSArray *items = [[BNRItemStore sharedStore] allItems];
    BNRItem *item = items[indexPath.row];
    cell.textLabel.text = item.description;

    // Configure the cell with the BNRItem
    cell.nameLabel.text = item.itemName;
    cell.serialNumberLabel.text = item.serialNumber;
    cell.valueLabel.text =
        [NSString stringWithFormat:@"%@", item.valueInDollars];

    return cell;
}
```

Primeiro, o identificador de reutilização é atualizado para refletir sua nova subclasse. O código no final desse método é bem óbvio: para cada rótulo na célula, determine sua `text` como alguma propriedade da `BNRItem` apropriada. (Você lidará com a `thumbnailView` mais tarde.)

Compile e execute o aplicativo e crie uma nova `BNRItem`. As células são carregadas, mas o layout delas provavelmente está desajustado. Você não definiu restrições para cada uma das subvisões da `contentView` da célula; então, faremos isso agora.

## Restrições para BNRItemCell

A visão de tabela da `BNRItemsViewController` mudará seu tamanho para corresponder ao tamanho da janela. Quando uma visão de tabela muda sua largura, cada uma das células também muda a largura para corresponder a ela. Assim, você deve configurar restrições na célula que reflitam essa mudança na largura. (A altura de uma célula não mudará a não ser que você peça explicitamente à visão de tabela que a mude, seja através da propriedade `rowHeight` ou de seu método delegate `tableView:heightForRowAtIndexPath:`.)

No momento, o `BNRItemCell.xib` tem posição e tamanho iniciais para cada visão. Por exemplo, o tamanho da `thumbnailView` no arquivo XIB é de 40 pontos de largura e 40 pontos de altura (se a sua não é exatamente 40x40, não se preocupe; terá esse tamanho em breve). Contudo, para o Auto Layout não importa o tamanho de uma visão quando ela é criada pela primeira vez; ele só se importa com as restrições. Se você adicionar uma restrição à visão de imagem que mantém sua largura em 500 pontos, a largura será de 500 pontos: o tamanho original não tem influência nisso.

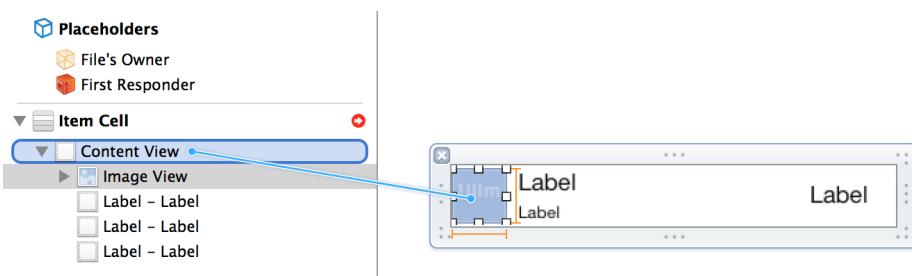
Estas são as restrições de que você precisa:

1. Fazer com que a `UIImageView` tenha um tamanho de 40x40 pontos, próximo à borda esquerda da visão de conteúdo e centralizada verticalmente com ela.
2. Certifique-se de que tanto a `nameLabel` quanto a `serialNumberLabel` tenham a mesma distância fixa da visão de imagem, se esticarem para preencher o comprimento da célula (menos o tamanho da visão de imagem e da `valueLabel`) e mantenham seu empilhamento vertical.
3. Mantenha a `valueLabel` centralizada verticalmente com a visão de conteúdo no lado direito da célula e a uma distância fixa dos outros dois rótulos.

Primeiro, fixe a largura e a altura de `imageView`. Você pode utilizar o menu Pin ou pressionar Control e arrastar diagonalmente da visão de imagem até ela mesma.

A seguir, centralize a visão de imagem dentro de seu recipiente. Você pode usar o menu Align para fazer isso (selecionando Vertical Center in container) ou pressionar Control e arrastar da visão de imagem para o recipiente (selecionando Center Vertically in Container). Se você pressionar Control e arrastar, tome cuidado para não arrastar para outra subvisão. Uma forma fácil para garantir que você arraste para a visão certa é arrastar para a Content View na estrutura do documento (Figure 19.7).

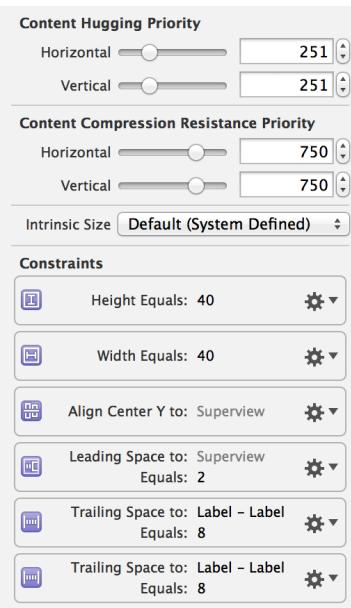
Figure 19.7 Arrastando para a estrutura do documento



Vamos configurar as restrições horizontais de todas as visões ao mesmo tempo. Selecione as quatro subvisões e, a partir do menu Pin, verifique os struts da esquerda e da direita na parte superior. Clique em Add 6 Constraints. A visão de imagem agora tem todas as restrições necessárias, como evidenciado pelas linhas de restrição azuis.

(Se as suas ainda não estão azuis, não se preocupe. O frame da sua visão de imagem pode não corresponder às restrições, e você consertará isso em breve.) As restrições concluídas da visão de imagem são mostradas na Figure 19.8.

Figure 19.8 Restrições da visão de imagem



Agora, conclua as restrições de nameLabel e serialNumberLabel. Selecione esses dois rótulos e abra o menu Pin. Verifique o strut de cima e de baixo, assim como Height, e clique em Add 5 Constraints. As restrições desses dois rótulos ficarão azuis, mas poderia haver restrições insatisfatórias se a altura da célula da visão de tabela nunca mudar. Por quê? No momento, todas as restrições verticais adicionadas têm sua igualdade definida como Equal. Na linguagem de formatação visual, isso dá a equação:

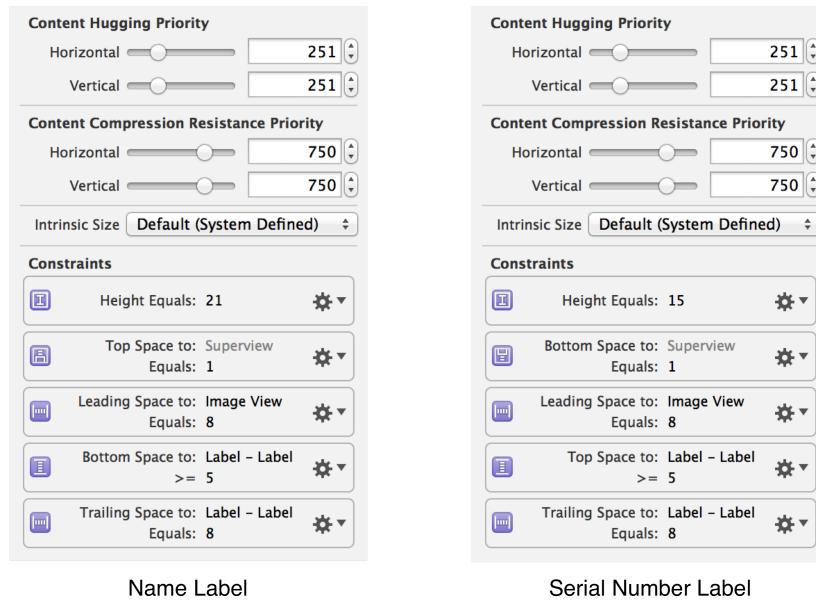
V: |-1-[nameLabel(==21)]-5-[serialNumberLabel(==15)]-1-|

Observe que esses valores, somados, resultam em 43, exatamente a altura atual da visão de conteúdo. Se a altura da célula da visão de tabela mudar, uma dessas restrições terá que ser quebrada. (Quer ver isso acontecer? Vá em frente e mude a altura da célula a partir do Size Inspector. Provavelmente seja uma boa ideia mudar de volta depois de você ver os efeitos, mas não é absolutamente necessário.) Você irá corrigir isso alterando a relação de uma das restrições para greater than or equal (maior ou igual).

A restrição que você irá modificar é a que fixa a parte inferior da nameLabel na parte superior da serialNumberLabel. Embora você possa selecionar essa restrição no canvas, ela é muito pequena e difícil de ser selecionada. Em vez disso, selecione nameLabel e abra seu Size inspector.

Aqui você pode ver todas as restrições que afetam a visão selecionada. Clique no ícone de engrenagem associado à restrição Bottom Space. Isso fará aparecer um menu que permitirá que você Select and Edit... (selecione e edite) ou Delete (exclua) a restrição. Escolha Select and Edit.... No topo do Attributes inspector, mude a Relation para Greater Than or Equal. As restrições desses dois rótulos são mostradas na Figure 19.9.

Figure 19.9 Restrições de nome e número de série do rótulo



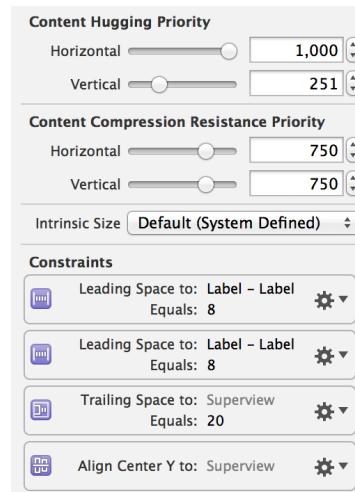
A seguir, selecione `valueLabel` e selecione e adicione a restrição para Vertical Center in Container no menu Align.

Você deve fazer uma última mudança para se livrar de um layout ambíguo. Já que nenhum dos três rótulos tem suas larguras fixadas, todas elas querem ter a largura de sua `intrinsicContentSize`. Já que a largura dos dois rótulos no meio + o espaçamento + a largura do rótulo de valor é maior do que suas larguras intrínsecas + esse espaçamento, algum deles terá que ceder: ou a largura de `nameLabel` e `serialNumberLabel` terá que aumentar, ou a largura de `valueLabel` terá que aumentar.

Vamos corrigir essa ambiguidade ajustando a prioridade de Content Hugging Priority do rótulo de valor para ser mais alta do que a dos outros dois rótulos. É uma abordagem melhor do que fixar a largura do rótulo de valor. Se a largura estiver fixa, então um texto maior do que o que pode ser exibido será truncado. Aumentando o Content Hugging Priority, a largura do rótulo de valor será sempre exatamente a largura necessária para exibir todo o texto. (A não ser que o rótulo seja tão longo que o texto deve ser truncado para satisfazer a todas as restrições.)

Selecione `valueLabel` e abra o Size Inspector. Altere Horizontal Content Hugging Priority para 1000. As restrições de `valueLabel` concluídas são mostradas na Figure 19.10.

Figure 19.10 Restrições do rótulo de valor



Você terminou! Todas as subvisões devem ter linhas de restrição azuis. Se alguma delas não tiver, selecione a célula e então clique em **Update All Frames in Item Cell** no menu **Resolve Auto Layout Issues**.

Deu um pouco de trabalho configurar a célula, mas seus conteúdos agora vão ser redimensionados de forma elegante se o tamanho da célula ou o conteúdo textual for alterado.

## Manipulação de imagens

Agora vamos abordar o conteúdo da **thumbnailView** na **BNRItemCell**. Para mostrar uma imagem dentro de uma célula, você pode redimensionar a imagem grande do item a partir do armazenamento de imagens. Entretanto, fazer isso prejudicaria o desempenho porque muitos bytes teriam que ser lidos, filtrados e redimensionados para caber na célula. Uma ideia melhor seria, em vez disso, criar e usar uma miniatura da imagem.

Para criar uma miniatura de uma imagem de **BNRItem**, você vai arrastar uma versão reduzida da imagem original para um contexto offscreen e manter uma referência para essa nova imagem dentro de uma instância de **BNRItem**. Você também precisa de um local para armazenar essa imagem em miniatura, para que ela possa ser recarregada quando o aplicativo for aberto novamente.

No Chapter 11, colocamos as imagens em tamanho original na **BNRImageStore** para que seja possível apagá-las da memória se necessário. Contudo, as imagens em miniatura serão pequenas o bastante para que você possa arquivá-las com as outras propriedades da **BNRItem**.

Abra o **BNRItem.h**. Declare uma nova propriedade para uma miniatura e um novo método que a configurará.

```
@property (nonatomic, copy) NSString *itemKey;
@property (nonatomic, strong) UIImage *thumbnail;

- (void)setThumbnailFromImage:(UIImage *)image;

@end
```

Quando uma imagem é escolhida para uma **BNRItem**, você entrega essa imagem à **BNRItem**. Ela reduzirá a imagem a um tamanho bem menor e manterá a imagem pequena como a sua **thumbnail** (miniatura).

O método que fará isso é o **setThumbnailFromImage:**. Ele pegará uma imagem de tamanho original, criará uma representação menor dela em um objeto de contexto gráfico offscreen e definirá a propriedade **thumbnail** como a imagem produzida pelo contexto offscreen. (Se você não sabe o que é um objeto de contexto gráfico, leia a the section called “Para os mais curiosos: Core Graphics” no Chapter 4).

O iOS fornece um pacote conveniente de funções para criar contextos offscreen e produzir imagens a partir deles. Para criar um contexto de imagem offscreen, você deve usar a função **UIGraphicsBeginImageContextWithOptions**. Essa função aceita uma estrutura **CGSize**, que especifica a largura e a altura do contexto de imagem, um fator de escala, e se a imagem será opaca. Quando essa função é chamada, uma nova **CGContextRef** é criada e se torna o contexto atual.

Para desenhar uma **CGContextRef**, você usa o Core Graphics, exatamente como se estivesse implementando um método **drawRect:** para uma subclasse **UIView**. Para obter uma **UIImage** a partir do contexto após ela ter sido desenhada, você invoca a função **UIGraphicsGetImageFromCurrentImageContext**.

Depois de produzir uma imagem a partir de um contexto de imagem, você deve limpar o contexto usando a função **UIGraphicsEndImageContext**.

No **BNRItem.m**, implemente os métodos a seguir para criar uma miniatura usando um contexto offscreen.

```

- (void)setThumbnailFromImage:(UIImage *)image
{
    CGSize origImageSize = image.size;
    // The rectangle of the thumbnail
    CGRect newRect = CGRectMake(0, 0, 40, 40);

    // Figure out a scaling ratio to make sure we maintain the same aspect ratio
    float ratio = MAX(newRect.size.width / origImageSize.width,
                      newRect.size.height / origImageSize.height);

    // Create a transparent bitmap context with a scaling factor
    // equal to that of the screen
    UIGraphicsBeginImageContextWithOptions(newRect.size, NO, 0.0);

    // Create a path that is a rounded rectangle
    UIBezierPath *path = [UIBezierPath bezierPathWithRoundedRect:newRect
                                                          cornerRadius:5.0];

    // Make all subsequent drawing clip to this rounded rectangle
    [path addClip];

    // Center the image in the thumbnail rectangle
    CGRect projectRect;
    projectRect.size.width = ratio * origImageSize.width;
    projectRect.size.height = ratio * origImageSize.height;
    projectRect.origin.x = (newRect.size.width - projectRect.size.width) / 2.0;
    projectRect.origin.y = (newRect.size.height - projectRect.size.height) / 2.0;

    // Draw the image on it
    [image drawInRect:projectRect];

    // Get the image from the image context; keep it as our thumbnail
    UIImage *smallImage = UIGraphicsGetImageFromCurrentImageContext();
    self.thumbnail = smallImage;

    // Cleanup image context resources; we're done
    UIGraphicsEndImageContext();
}

```

No `BNRDetailViewController.m`, adicione a linha de código a seguir ao `imagePickerController:didFinishPickingMediaWithInfo:`: para criar uma miniatura quando a câmera fotografar a imagem original.

```

- (void)imagePickerController:(UIImagePickerController *)picker
didFinishPickingMediaWithInfo:(NSDictionary *)info
{
    NSString *oldKey = self.item.itemKey;

    if (oldKey) {
        [[BNRImageStore sharedStore] deleteImageForKey:oldKey];
    }

    UIImage *image = info[UIImagePickerControllerOriginalImage];
    [self.item setThumbnailFromImage:image];
}

```

Agora que as instâncias de `BNRItem` têm uma miniatura, você pode usar essa miniatura na visão de tabela de `BNRItemsViewController`. No `BNRItemsViewController.m`, atualize o `tableView:cellForRowAtIndexPath:`:

```

cell.valueLabel.text =
    [NSString stringWithFormat:@"%@", item.valueInDollars];

cell.thumbnailView.image = item.thumbnail;

return cell;
}

```

Agora, compile e execute o aplicativo. Tire uma fotografia para uma instância de `BNRItem` e retorne para a visão de tabela. Essa linha exibirá uma imagem em miniatura junto com o nome e o valor da `BNRItem`. (Observe que você terá que tirar novamente as fotos de itens existentes.)

Não se esqueça de adicionar os dados das miniaturas em seu arquivo! Abra o `BNRItem.m` e faça as seguintes alterações:

```
- (id)initWithCoder:(NSCoder *)aDecoder
{
    self = [super init];

    if (self) {
        _itemName = [aDecoder decodeObjectForKey:@"itemName"];
        _serialNumber = [aDecoder decodeObjectForKey:@"serialNumber"];
        _dateCreated = [aDecoder decodeObjectForKey:@"dateCreated"];
        _itemKey = [aDecoder decodeObjectForKey:@"itemKey"];
        _thumbnail = [aDecoder decodeObjectForKey:@"thumbnail"];

        _valueInDollars = [aDecoder decodeIntForKey:@"valueInDollars"]
    }
    return self;
}

- (void)encodeWithCoder:(NSCoder *)aCoder
{
    [aCoder encodeObject:self.itemName forKey:@"itemName"];
    [aCoder encodeObject:self.serialNumber forKey:@"serialNumber"];
    [aCoder encodeObject:self.dateCreated forKey:@"dateCreated"];
    [aCoder encodeObject:self.itemKey forKey:@"itemKey"];
    [aCoder encodeObject:self.thumbnail forKey:@"thumbnail"];

    [aCoder encodeInt:self.valueInDollars forKey:@"valueInDollars"];
}
```

Compile e execute o aplicativo. Tire algumas fotos de itens e depois saia e reabra o aplicativo. As miniaturas aparecerão agora para os itens salvos.

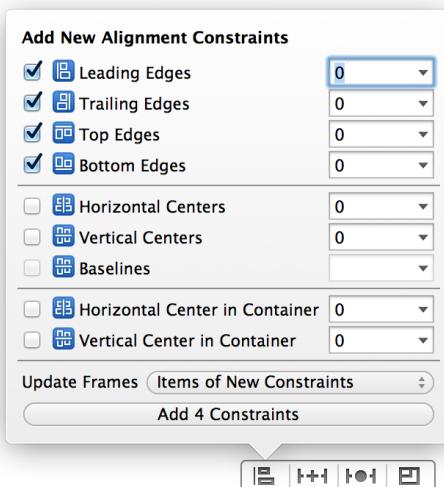
## Retransmissão de ações de UITableViewCells

Às vezes, pode ser útil adicionar uma **UIControl** ou uma de suas subclasses, como uma **UIButton**, a uma **UITableViewCell**. Por exemplo, você quer que os usuários consigam tocar a miniatura em uma célula e ver uma imagem do item em tamanho original. Nesta seção, você fará isso adicionando um botão transparente em cima da miniatura. Tocar nesse botão exibirá a imagem em tamanho original em uma **UIPopoverController** quando o aplicativo estiver rodando em um iPad.

Abra o `BNRItemCell.m` e apague um método que acionará a exibição da imagem.

```
- (IBAction)showImage:(id)sender
{
}
```

Agora, no `BNRItemCell.xib`, arraste uma **UIButton** para a visão de conteúdo. Remova o texto do botão. Selecione a **UIImageView** e a **UIButton**, e então, no menu de Auto Layout Align, selecione Leading Edges, Trailing Edges, Top Edges e Bottom Edges. Para o menu suspenso Update Frames, selecione Items of New Constraints e então clique em Add 4 Constraints (Figure 19.11).

Figure 19.11 Restrições de **UIButton**

Finalmente, pressione Control e arraste de **UIButton** para Item Cell e conecte-a ao método **showImage:**.

Agora você tem um botão que enviará **showImage:** para **BNRItemCell** sempre que for pressionado. Obviamente, você terá de implementar esse método, mas aqui você se depara com um problema: essa mensagem será enviada para a **BNRItemCell**, mas **BNRItemCell** não é um controlador e não tem acesso a nenhum dos dados de imagem necessários para obter a imagem em tamanho original. Na verdade, ela não tem nem acesso à **BNRItem** cuja miniatura está exibindo.

Você poderia, talvez, deixar a **BNRItemCell** manter um ponteiro para a **BNRItem** que ela exibe. Mas as células da visão de tabela são objetos de visão, e não devem gerenciar objetos de modelo, nem conseguir apresentar interfaces adicionais (como a **UIPopoverController**).

Uma solução melhor seria dar a **BNRItemCell** um bloco para executar quando o botão for pressionado. Esse bloco será fornecido pela **BNRItemsViewController** que é responsável pela configuração da célula.

## Adição de um bloco à subclasse da célula

Analisamos os blocos brevemente no Chapter 17, mas vamos fazer uma análise mais detalhada agora.

Abra a **BNRItemCell.h** e adicione uma propriedade para o bloco que será executado.

```
@interface BNRItemCell : UITableViewCell

@property (nonatomic, weak) IBOutlet UIImageView *thumbnailView;
@property (nonatomic, weak) IBOutlet UILabel *nameLabel;
@property (nonatomic, weak) IBOutlet UILabel *serialNumberLabel;
@property (nonatomic, weak) IBOutlet UILabel *valueLabel;

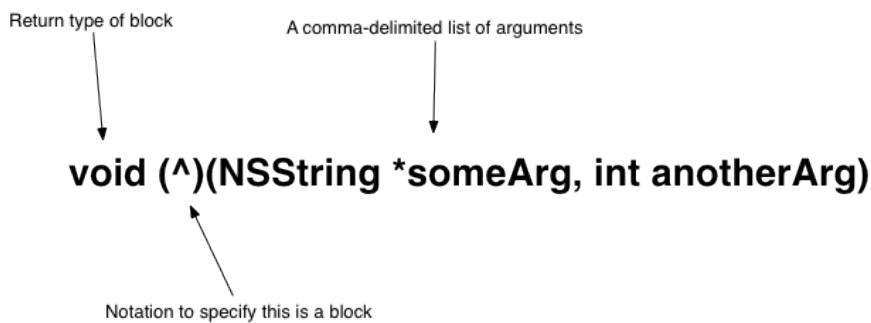
@property (nonatomic, copy) void (^actionBlock)(void);

@end
```

A sintaxe pode ser ainda um pouco assustadora, mas lembre-se de que um bloco parece muito com uma função. A Figure 19.12 mostra os vários componentes de um bloco.

Observe que a propriedade é declarada como **copy**. Isso é muito importante. Os blocos comportam-se de uma forma um pouco diferente do restante dos objetos com os quais você trabalhou até agora. Quando um bloco é criado, ele é criado na pilha, ao invés de ser criado no heap como outros objetos. Isso significa que, quando o método no qual um bloco é declarado é retornado, quaisquer blocos que foram criados serão destruídos juntamente com todas as outras variáveis locais. Para que um bloco persista além do método no qual ele é declarado, ele precisa receber a mensagem **copy**. Ao fazer isso, o bloco será copiado para o heap – assim, você declara que a propriedade tenha o atributo **copy**.

Figure 19.12 Sintaxe de blocos



No `BNRItemCell.m`, chame o bloco quando o botão for pressionado.

```
- (IBAction)showImage:(id)sender
{
    if (self.actionBlock) {
        self.actionBlock();
    }
}
```

Observe que você tem que verificar se o bloco existe antes de chamá-lo.

Vamos verificar se tudo isso funciona conforme planejado. No `BNRItemsViewController.m`, atualize o `tableView:cellForRowAtIndexPath:` para imprimir o caminho índice.

```
- (UITableViewCell *)tableView:(UITableView *)tableView
    cellForRowAtIndexPath:(NSIndexPath *)indexPath
{
    BNRItem *item = [[BNRItemStore sharedStore] allItems][indexPath.row];

    // Get the new or recycled cell
    BNRItemCell *cell =
        [tableView dequeueReusableCellWithIdentifier:@"BNRItemCell"
                                         forIndexPath:indexPath];

    // Configure the cell with the BNRItem
    cell.nameLabel.text = item.itemName;
    cell.serialNumberLabel.text = item.serialNumber;
    cell.valueLabel.text = [NSString stringWithFormat:@"%@", item.valueInDollars];
    cell.thumbnailView.image = item.thumbnail;

    cell.actionBlock = ^{
        NSLog(@"Going to show image for %@", item);
    };

    return cell;
}
```

Compile e execute o aplicativo. Toque em uma miniatura (ou, mais precisamente, no botão transparente em cima da miniatura) e verifique a mensagem no console.

## Apresentação de imagens em um controlador popover

Agora, a `BNRItemsViewController` precisa mudar o bloco de ação para capturar a `BNRItem` associada à célula cujo botão foi tocado, e exibir sua imagem em um `UIPopoverController`.

Para mostrar uma imagem em um popover, você precisa de uma nova `UIViewController` cuja view mostra uma imagem. Crie uma nova subclasse de Objective-C. Nomeie essa nova classe como `BNRImageViewController`, selecione a `UIViewController` como sua superclasse e desmarque todas as caixas.

Já que esse controlador de visão terá apenas uma visão, você o criará de forma programática. No `BNRImageViewController.m`, implemente `loadView`.

```
- (void)loadView
{
    UIImageView *imageView = [[UIImageView alloc] init];
    self.view = imageView;
}
```

Observe que você não precisa configurar quaisquer restrições, pois a `UIPopoverController` na qual a `BNRImageViewController` aparecerá sempre definirá o tamanho dessa visão de imagem como igual ao tamanho do popover.

Agora, adicione uma propriedade à interface pública no `BNRImageViewController.h` para armazenar a imagem.

```
@interface BNRImageViewController : UIViewController
@property (nonatomic, strong) UIImage *image;
@end
```

Quando uma instância de `BNRImageViewController` é criada, ela recebe uma imagem. No `BNRImageViewController.m`, implemente `viewWillAppear:` para configurar a imagem da view a partir dessa `image`.

```
- (void)viewWillAppear:(BOOL)animated
{
    [super viewWillAppear:animated];
    // We must cast the view to UIImageView so the compiler knows it
    // is okay to send it setImage:
    UIImageView *imageView = (UIImageView *)self.view;
    imageView.image = self.image;
}
```

Agora, você pode terminar de implementar o bloco de ação. No `BNRItemsViewController.m`, adicione uma propriedade para se fixar a um controlador popover na extensão de classe e fazer com que essa classe fique em conformidade com o protocolo `UIPopoverControllerDelegate`.

```
@interface BNRItemsViewController () <UIPopoverControllerDelegate>
@property (nonatomic, strong) UIPopoverController *imagePopover;
@end
```

Depois, importe os arquivos de cabeçalho apropriados no topo do `BNRItemsViewController.m`.

```
#import "BNRImageStore.h"
#import "BNRImageViewController.h"
```

Complete a implementação do bloco de ação para apresentar o controlador popover que exibe a imagem em tamanho original para a `BNRItem` representada pela célula que foi tocada.

```
cell.actionBlock = ^{
    NSLog(@"Going to show image for %@", item);

    if ([UIDevice currentDevice].userInterfaceIdiom == UIUserInterfaceIdiomPad) {

        NSString *itemKey = item.itemKey;

        // If there is no image, we don't need to display anything
        UIImage *img = [[BNRImageStore sharedStore] imageForKey:itemKey];
        if (!img) {
            return;
        }

        // Make a rectangle for the frame of the thumbnail relative to
        // our table view
        // Note: there will be a warning on this line that we'll soon discuss
        CGRect rect = [self.view convertRect:cell.thumbnailView.bounds
                                      fromView:cell.thumbnailView];

        // Create a new BNRImageViewController and set its image
        BNRImageViewController *ivc = [[BNRImageViewController alloc] init];
        ivc.image = img;

        // Present a 600x600 popover from the rect
        self.imagePopover = [[UIPopoverController alloc]
                             initWithContentViewController:ivc];
        self.imagePopover.delegate = self;
        self.imagePopover.popoverContentSize = CGSizeMake(600, 600);
        [self.imagePopover presentPopoverFromRect:rect
                                         inView:self.view
                                         permittedArrowDirections:UIPopoverArrowDirectionAny
                                         animated:YES];
    }
};
```

Finalmente, no `BNRItemsViewController.m`, livre-se do popover caso o usuário toque em qualquer lugar fora dele.

```
- (void)popoverControllerDidDismissPopover:(UIPopoverController *)popoverController
{
    self.imagePopover = nil;
}
```

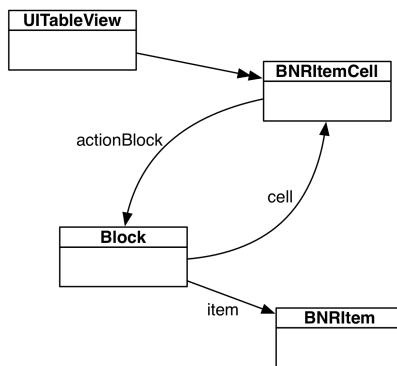
Compile e execute o aplicativo. Toque nas miniaturas em cada linha para ver a imagem em tamanho original no popover. Toque em qualquer outro lugar para dispensar o popover.

## Captura de variáveis

Um bloco pode usar quaisquer variáveis que estejam visíveis dentro de seu *escopo delimitador*. O escopo delimitador de um bloco é o escopo do método no qual ele é definido. Portanto, um bloco tem acesso a todas as variáveis locais do método, aos argumentos passados ao método e às variáveis de instância que pertencem ao objeto que executa o método. No código `actionBlock` acima, tanto a `BNRItem` (`item`) quanto a `BNRItemCell` (`cell`) foram capturadas a partir do escopo delimitador.

Os blocos possuem os objetos que eles capturaram, e isso pode resultar facilmente em um ciclo de referência forte. Volte a observar o aviso que você recebeu ao criar a `rect` dentro da `actionBlock`: “capturar ‘cell’ de forma forte neste bloco provavelmente vai levar a um ciclo de retenção”. Uma vez que os blocos possuem os objetos que capturam, isso faz sentido. A `cell` tem a propriedade de `actionBlock`, e a `actionBlock` tem uma forte propriedade de `cell` (Figure 19.13).

Figure 19.13 A célula e o bloco retém um ao outro



Para corrigir esse problema, a `actionBlock` deve ter uma referência fraca para `cell`, o que quebrará o ciclo de referência forte.

No `BNRItemsViewController.m`, atualize o código da `actionBlock` para fazer uma referência fraca à `BNRItemCell`.

```

weakSelf = self;
weakCell = cell;
cell.actionBlock = ^{
    NSLog(@"Going to show image for %@", item);
    BNRItemCell *strongCell = weakSelf;
    if ([UIDevice currentDevice].userInterfaceIdiom == UIUserInterfaceIdiomPad) {
        NSString *itemKey = item.itemKey;
        // If there is no image, we don't need to display anything
        UIImage *img = [[BNRImageStore sharedStore] imageForKey:itemKey];
        if (!img) {
            return;
        }
        // Make a rectangle that the frame of the thumbnail relative to
        // our table view
        // Note: there will be a warning on this line that we'll soon discuss
        // CGRect rect = [self.view convertRect:cell.thumbnailView.bounds
        //                     fromView:cell.thumbnailView];
        CGRect rect = [self.view convertRect:strongCell.thumbnailView.bounds
                     fromView:strongCell.thumbnailView];
        // Create a new BNRImageViewController and set its image
        BNRImageViewController *ivc = [[BNRImageViewController alloc] init];
        ivc.image = img;
        // Present a 600x600 popover from the rect
        self.imagePopover = [[UIPopoverController alloc]
                            initWithContentViewController:ivc];
        self.imagePopover.delegate = self;
        self.imagePopover.popoverContentSize = CGSizeMake(600, 600);
        [self.imagePopover presentPopoverFromRect:rect
                                         inView:self.view
                                         permittedArrowDirections:UIPopoverArrowDirectionAny
                                         animated:YES];
    }
};
    
```

Assim que o bloco iniciar a execução, você precisa garantir que a célula continue ali até a execução parar. Por esse motivo, você temporariamente toma propriedade forte daquela célula criando uma referência forte a ela com `strongCell`. Diferentemente de fazer uma referência forte permanente a uma variável a partir do escopo delimitador, desta forma o bloco só tem uma referência forte ao objeto de célula enquanto a variável `strongCell` existe, ou seja, enquanto o bloco realmente está sendo executado.

Compile e execute o aplicativo. O comportamento será o mesmo, mas o seu aplicativo não terá mais um vazamento de memória.

## Desafio de bronze: código de cores

Se uma **BNRItem** valer mais que \$ 50, faça com que o texto do rótulo de valor apareça em verde. Se valer menos que \$ 50, faça com que apareça em vermelho.

## Desafio de ouro: zoom

A **BNRImageViewController** deve centralizar sua imagem e permitir o uso de zoom. Implemente esse comportamento no **BNRImageViewController.m**.

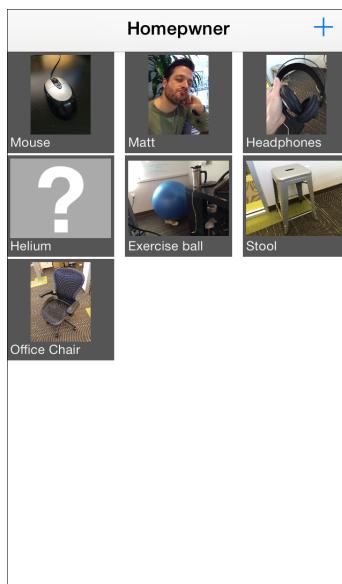
## Para os mais curiosos: UICollectionView

A classe **UICollectionView** é muito semelhante à **UITableView**:

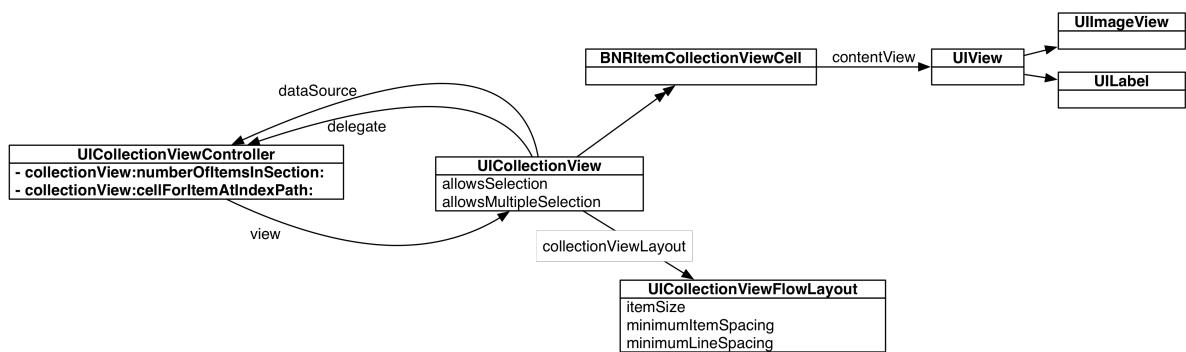
- Trata-se de uma subclasse de **UIScrollView**.
- Ela exibe células, embora essas células herdem de **UICollectionViewCell** em vez de **UITableViewCell**.
- Tem uma fonte de dados que fornece essas células para ela.
- Tem um delegate que se informa sobre coisas como a seleção de uma célula.
- Semelhante à **UIViewController**, **UICollectionViewController** é uma classe de controlador de visão que cria uma **UICollectionView** como sua visão e torna-se seu delegate e fonte de dados.

Em que a visão de coleção difere da visão de tabela? Uma visão de tabela só mostra uma coluna de células; isso é uma grande limitação em um dispositivo de tela grande como o iPad. Uma visão de coleção pode dispor as células da forma que você quiser. O layout mais comum é uma grade (Figure 19.14).

Figure 19.14 Homeowner com uma **UICollectionView**



Como a **UICollectionView** sabe como organizar as células? Ela tem um objeto de layout que controla os atributos de cada célula, incluindo sua posição e tamanho. Esses objetos de layout herdam da classe abstrata **UICollectionViewLayout**. Se você estiver apenas disposta suas células em uma grade, você pode utilizar uma instância de **UICollectionViewFlowLayout**. Se estiver fazendo algo mais sofisticado, você precisará criar uma subclasse personalizada de **UICollectionViewLayout**.

Figure 19.15 Exemplo de modelo de objeto para uma **UICollectionView**

A **UITableViewCell** padrão até que é bem utilizável. (Você a usou em vários capítulos anteriores.) A **UICollectionViewCell** não é. Ela tem uma visão de conteúdo, mas a visão de conteúdo não tem subvisões. Assim, se você estiver criando uma **UICollectionView**, precisará criar uma subclasse de **UICollectionViewCell**.

Isso é tudo o que você precisa saber para fazer funcionar a sua primeira visão de coleção. Depois disso, você irá querer brincar com a visão de fundo, visões suplementares (que são usadas basicamente como cabeçalhos e rodapés de seções) e visões decorativas. A célula também tem sua própria visão de fundo e uma visão de fundo selecionada (que é sobreposta à visão de fundo quando a célula é selecionada).



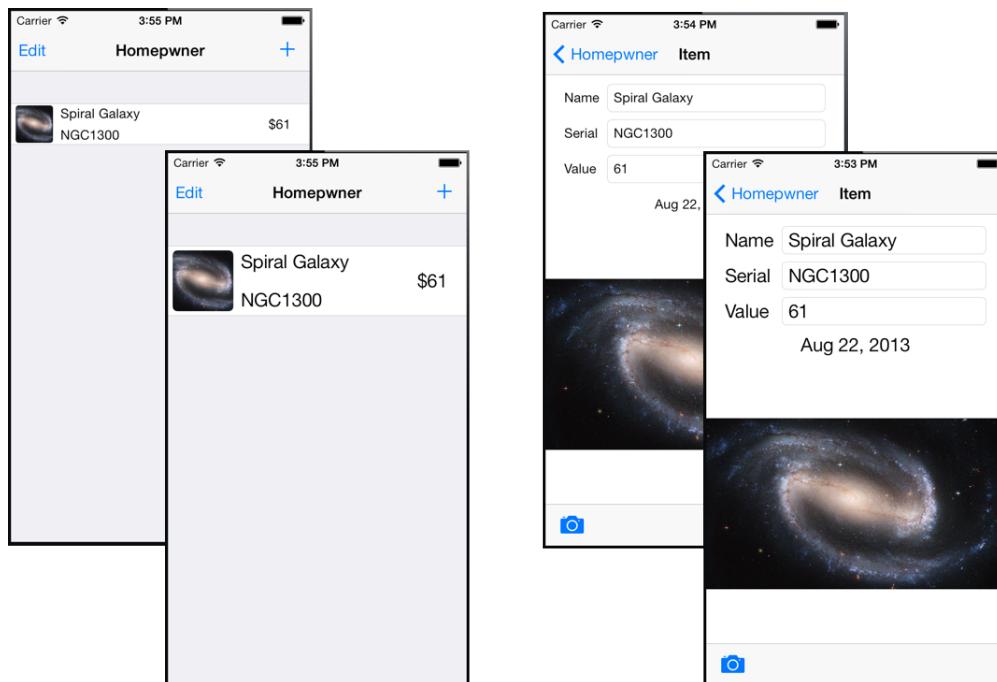
# 20

## Dynamic Type (tipo dinâmico)

Criar uma interface que agrade a todos pode ser uma tarefa difícil. Alguns preferem interfaces mais compactas, assim podem ver mais informações ao mesmo tempo. Outros querem ver as informações de forma fácil e rápida, ou talvez não tenham uma visão boa. Essas pessoas têm diferentes necessidades, e os bons desenvolvedores empenham-se para criar aplicativos que atendam a essas necessidades.

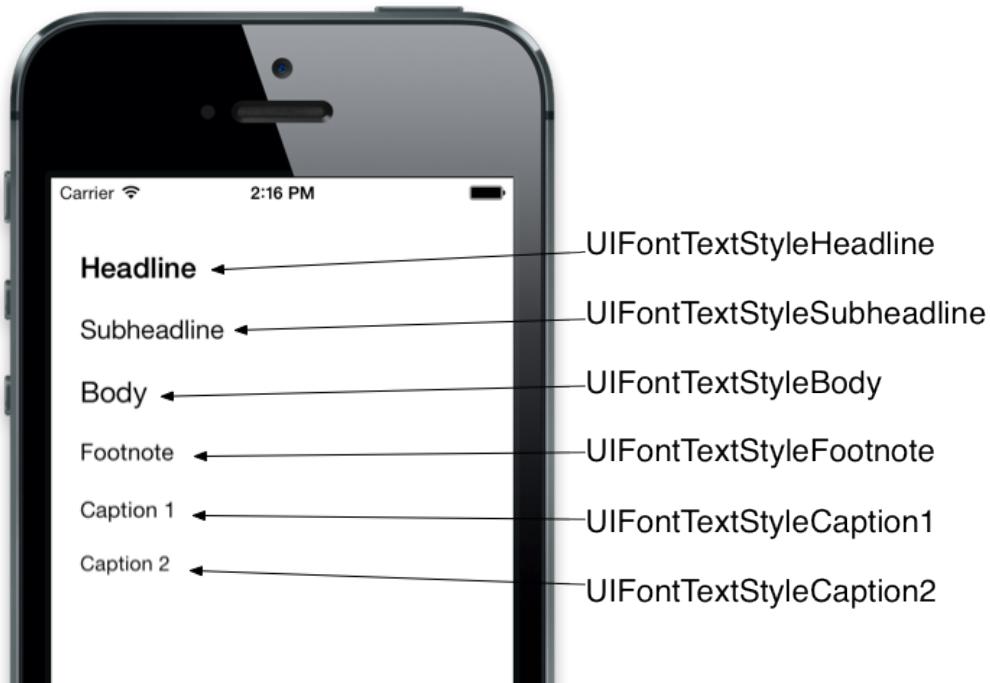
*Dynamic Type* é uma tecnologia lançada no iOS 7 que ajuda a alcançar esse objetivo oferecendo *estilos de texto* especificamente projetados, que são otimizados para legibilidade. Talvez ainda mais importante, os usuários podem selecionar no aplicativo *Settings* da Apple um dos sete diferentes tamanhos de texto preferenciais, e os aplicativos que suportam o Dynamic Type terão fontes dimensionadas de forma adequada. Neste capítulo, você atualizará o *Homepwner* para que seja compatível com o Dynamic Type. A Figure 20.1 mostra o aplicativo renderizado no menor e no maior tamanho de Dynamic Type (tipo dinâmico) selecionável pelo usuário.

Figure 20.1 Homepwner com Dynamic Type suportado



O sistema Dynamic Type centraliza-se em torno de *estilos de texto*. Quando uma fonte for solicitada por um determinado estilo de texto, o sistema utilizará o tamanho de texto preferencial do usuário em associação com o estilo de texto para retornar uma fonte apropriadamente configurada. A Figure 20.2 mostra os seis diferentes estilos de texto.

Figure 20.2 Diferentes estilos de texto



## Utilização de fontes preferenciais

A implementação do Dynamic Type é direta. No nível mais básico, você obtém uma **UIFont** para um estilo de texto específico e então aplica essa fonte a alguma coisa que exiba textos, como a **UILabel**. Vamos começar atualizando **BNRDetailViewController**.

Em breve, você precisará atualizar alguns atributos dos rótulos de forma programática, e também adicionar outlets em cada um dos rótulos da extensão de classe no **BNRDetailViewController.m**.

```
@interface BNRDetailViewController : UIViewController

@property (nonatomic, strong) UIPopoverController *imagePickerPopover;

@property (weak, nonatomic) IBOutlet UITextField *nameField;
@property (weak, nonatomic) IBOutlet UITextField *serialNumberField;
@property (weak, nonatomic) IBOutlet UITextField *valueField;
@property (weak, nonatomic) IBOutlet UILabel *dateLabel;
@property (weak, nonatomic) IBOutlet UIImageView *imageView;
@property (weak, nonatomic) IBOutlet UIToolbar *toolbar;
@property (weak, nonatomic) IBOutlet UIBarButtonItem *cameraButton;

@property (weak, nonatomic) IBOutlet UILabel *nameLabel;
@property (weak, nonatomic) IBOutlet UILabel *serialNumberLabel;
@property (weak, nonatomic) IBOutlet UILabel *valueLabel;

@end
```

Agora que há um outlet em cada um dos rótulos, adicione um método que defina a fonte para que cada um use o estilo de corpo preferencial.

```

- (void)updateFonts
{
    UIFont *font = [UIFont preferredFontForTextStyle:UIFontTextStyleBody];

    self.nameLabel.font = font;
    self.serialNumberLabel.font = font;
    self.valueLabel.font = font;
    self.dateLabel.font = font;

    self.nameField.font = font;
    self.serialNumberField.font = font;
    self.valueField.font = font;
}

```

Agora, chame esse método no final de **viewWillAppear:** para atualizar os rótulos antes que eles se tornem visíveis.

```

// Clear the imageView
self.imageView.image = nil;
}

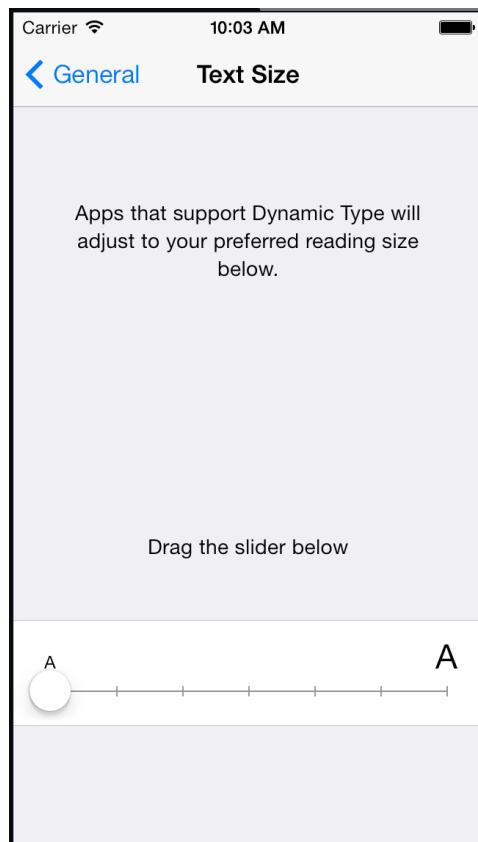
[self updateFonts];
}

```

O método **preferredFontForTextStyle:** retornará uma fonte pré-configurada, personalizada de acordo com as preferências do usuário. Compile e execute o aplicativo. Você perceberá que a interface ficará muito parecida.

Agora, vamos alterar o tamanho da fonte preferencial. Pressione o botão Home (ou use Home no menu Hardware) e abra o aplicativo Settings da Apple. Em General, selecione Text Size, e arraste o controle deslizante totalmente para a esquerda para configurar o tamanho da fonte com o menor valor (Figure 20.3).

Figure 20.3 Configurações de tamanho de texto



Agora, vamos voltar para o Homeowner. Se você retornar à **BNRDetailViewController**, perceberá que a interface não mudou em nada! Por que isso acontece? Como o **viewWillAppear:** não é chamado quando o

aplicativo retorna do segundo plano, sua interface não é atualizada. Felizmente, você pode ser avisado quando o usuário mudar o tamanho de fonte preferencial.

## Respondendo a alterações do usuário

Quando o usuário altera o tamanho do texto preferencial, uma notificação é gerada, informando que os objetos do aplicativo podem se registrar para detecção. Trata-se de `UIContentSizeCategoryDidChangeNotification`. E este é um ótimo momento para atualizar a interface do usuário.

No `BNRDetailViewController.m`, registre-se para essa notificação no `initForNewItem:` e remova a classe como observador no `dealloc`.

```
    self.navigationItem.leftBarButtonItem = cancelItem;
}

// Make sure this is NOT in the if (isNew) { } block of code
NSNotificationCenter *defaultCenter = [NSNotificationCenter defaultCenter];
[defaultCenter addObserver:self
    selector:@selector(updateFonts)
    name:UIContentSizeCategoryDidChangeNotification
    object:nil];
}

return self;
}

- (void)dealloc
{
    NSNotificationCenter *defaultCenter = [NSNotificationCenter defaultCenter];
    [defaultCenter removeObserver:self];
}
```

Observe que o método que está sendo chamado pela central de notificações é o mesmo implementado anteriormente, que também é chamado no `viewWillAppear:`. Compile e execute o aplicativo novamente; a interface será atualizada assim que você alterar o tamanho do texto preferencial do tipo dinâmico em `Settings`. Os rótulos e campos de texto na `BNRDetailViewController` serão dimensionados com estilos nas preferências do usuário.

É só isso? Sim, basicamente. Agora que sua interface pode crescer e encolher de forma dinâmica, é preciso rever as restrições do Auto Layout (layout automático).

## Atualização do Auto Layout

Quando as restrições foram configuradas para os rótulos `Name`, `Serial Number` e `Value`, você fixou a largura e a altura. Tudo funcionava bem quando o texto era uma fonte e um tamanho de texto fixo, mas agora que você conheceu o Dynamic Type (tipo dinâmico), sua interface não é dimensionada. Se o usuário seleciona um tamanho de texto pequeno, sobra muito espaço em branco vazio; se o usuário seleciona um tamanho de texto muito grande, o texto pode ficar cortado. O que você precisa fazer é utilizar a `intrinsicContentSize` dos rótulos para permitir que eles se redimensionem sozinhos, ficando exatamente do tamanho que precisam ficar.

Abra o `BNRDetailViewController.xib`. No canvas, selecione cada um dos quatro rótulos, um por um, e remova as restrições explícitas de largura e altura. Se você possui alguma visão colocada no lugar errado, selecione `UIControl` no canvas e `Update All Frames in Control` no menu `Resolve Auto Layout Issues`.

Se você olhar mais de perto, ou se você alterar temporariamente o texto de um dos rótulos para que fique um pouco menor ou maior, você verá que os campos de texto não ficam mais alinhados. Eles ficariam melhores se estivessem alinhados, mas antes de descobrir uma solução, entenda como o Auto Layout está calculando todos os frames.

## Revendo envolvimento de conteúdo e prioridades de resistência à compressão

Lembre-se de que cada visão possui um tamanho preferencial, ou seja, sua `intrinsicContentSize`. Elas concedem a cada visão uma restrição implícita de largura e altura. Para que uma visão fique maior do que sua

`intrinsicContentSize` em determinada dimensão, é necessário que haja uma restrição com prioridade mais alta do que a prioridade de envolvimento de conteúdo dessa visão. Para que uma visão fique menor do que sua `intrinsicContentSize` em determinada dimensão, é necessário que haja uma restrição com prioridade mais alta do que a prioridade de resistência à compressão de conteúdo dessa visão.

É importante lembrar-se dessas informações ao determinar como as interfaces serão dispostas. Vamos analisar o layout do rótulo Name e o campo de texto correspondente na direção horizontal. As restrições que afetam essas visões são:

- `nameLabel.leadingAnchor =Superview.leadingAnchor + 8`
- `nameField.leadingAnchor = nameLabel.trailingAnchor + 8`
- `nameField.trailingAnchor =Superview.trailingAnchor - 8`

Isso nos dá uma string de formatação visual semelhante a:

H: |-8-[`nameLabel`]-8-[`nameField`]-8- |

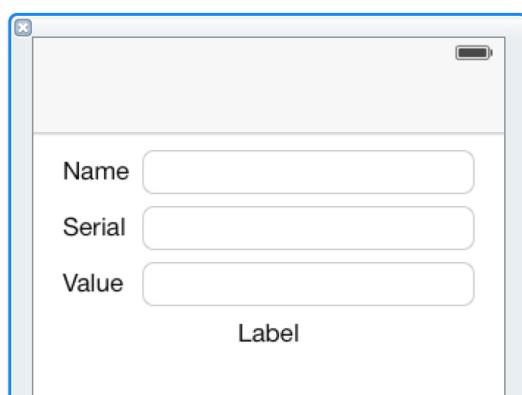
Observe que não há restrições que causem impacto direto na largura das visões. Por esse motivo, ambas as visões desejam ficar com a mesma largura de sua `intrinsicContentSize`. Um ou ambos os rótulos terão de se esticar para satisfazer as restrições existentes.

Portanto, que visão será esticada? Uma vez que ambas as visões desejam ficar mais largas que sua `intrinsicContentSize`, a visão com a menor prioridade de envolvimento de conteúdo será esticada. Se você comparar a **UILabel** e a **UITextField**, você verá que o rótulo possui uma prioridade de envolvimento de conteúdo de 251, e a do campo de texto é de 250. Já que o rótulo deseja “envolver” mais, ele ficará com a mesma largura de sua `intrinsicContentSize`, e o campo de texto será esticado o suficiente para satisfazer as equações.

Lembre-se de que o objetivo é que todos os campos de texto fiquem alinhados. A maneira como você consegue isso é fazendo com que os três rótulos superiores fiquem com a mesma largura. Isso pode parecer aquilo que você acabou de remover, mas há uma diferença sutil: anteriormente, cada um dos rótulos tinha largura (e altura) fixa independentemente, mas agora eles terão sempre larguras iguais.

Selecione todos os rótulos Name, Serial e Value juntos e abra o menu Pin. Selecione Equal Widths e no menu suspenso Update Frames, selecione All Frames in Container. Por fim, clique em Add 2 Constraints. Sua interface deve ser semelhante à da Figure 20.4.

Figure 20.4 Adição de restrições de larguras iguais



Até que foi fácil conseguir esse resultado, mas você pode estar se perguntando como tudo isso vai funcionar. Vamos dar uma olhada mais a fundo, recapitular algumas informações anteriores. Primeiro, a prioridade de envolvimento de conteúdo de cada rótulo (251) é maior do que de seu campo de texto correspondente (250), portanto, o rótulo fará o envolvimento antes do campo de texto. Por fim, você simplesmente adicionou restrições de larguras iguais para os três rótulos superiores.

Portanto, cada um dos três rótulos superiores possui duas restrições que causam impacto na largura: uma restrição obrigatória (prioridade 1000) Equal Widths, e duas restrições implícitas que tentam manter o rótulo em sua intrinsicContentSize. (Envolvimento de conteúdo com prioridade 251, e resistência à compressão de conteúdo com prioridade 750). Cada uma das visões deseja satisfazer todas essas restrições. A única visão que poderá fazer isso, contudo, será aquela que tiver a largura máxima. As outras duas visões terão uma restrição obrigatória com prioridade mais alta do que a prioridade de envolvimento de conteúdo, portanto, elas se esticarão para que tenham a mesma largura do rótulo maior.

O que aconteceu aqui é muito importante. Você criou uma interface que é dimensionada perfeitamente de acordo com as alterações de conteúdo do texto. As alterações de texto podem ocorrer por inúmeros motivos diferentes. É mais comum que isso ocorra por causa do uso do tipo dinâmico ou da localização do aplicativo para idiomas diferentes (que abordaremos no Chapter 25).

A interface da **BNRDetailViewController** está pronta. Agora, basta testar a **BNRDetailViewController** em diferentes tamanhos de tipo dinâmico, e você perceberá que a interface é redimensionada apropriadamente.

## Determinação do tamanho do texto preferencial do usuário

Agora é o momento de nos atermos à **BNRItemsViewController**. Você precisará atualizar duas partes desse controlador de visão para Dynamic Type: as linhas de sua visão de tabela crescerão ou encolherão em resposta à alteração do tamanho de texto preferencial feita pelo usuário, e **BNRItemCell** terá de ser atualizada de forma semelhante à atualização de **BNRDetailViewController**. Vamos começar atualizando a altura da linha da visão de tabela.

O objetivo é fazer com que a altura da linha da visão de tabela reflita o tamanho do texto preferencial do tipo dinâmico do usuário. Se o usuário optar por um tamanho de texto maior, as linhas ficarão mais altas para acomodar o texto. Como esse não é um problema que o Auto Layout resolve, a altura das linhas terá de ser definida manualmente. Para isso, você precisará de uma forma para determinar que tamanho de texto o usuário selecionou.

**UIApplication** expõe o tamanho do texto que o usuário selecionou através de sua propriedade `preferredContentSizeCategory`. O método retornará uma constante **NSString** com o nome da categoria de tamanho do conteúdo, que será um dos seguintes valores:

- `UIContentSizeCategoryExtraSmall`
- `UIContentSizeCategorySmall`
- `UIContentSizeCategoryMedium`
- `UIContentSizeCategoryLarge` (padrão)
- `UIContentSizeCategoryExtraLarge`
- `UIContentSizeCategoryExtraExtraLarge`
- `UIContentSizeCategoryExtraExtraExtraLarge`

Abra o `BNRItemsViewController.m`. Crie um método que atualizará a altura da linha da visão de tabela com base no tamanho do texto selecionado pelo usuário e chame esse método no `viewWillAppear:`.

```

- (void)viewWillAppear:(BOOL)animated
{
    [super viewWillAppear:animated];
    [self.tableView reloadData];

    [self updateTableViewForDynamicTypeSize];
}

- (void)updateTableViewForDynamicTypeSize
{
    static NSDictionary *cellHeightDictionary;

    if (!cellHeightDictionary) {
        cellHeightDictionary = @{@"UIContentSizeCategoryExtraSmall" : @44,
                               @"UIContentSizeCategorySmall" : @44,
                               @"UIContentSizeCategoryMedium" : @44,
                               @"UIContentSizeCategoryLarge" : @44,
                               @"UIContentSizeCategoryExtraLarge" : @55,
                               @"UIContentSizeCategoryExtraExtraLarge" : @65,
                               @"UIContentSizeCategoryExtraExtraExtraLarge" : @75 };
    }

    NSString *userSize =
        [[UIApplication sharedApplication] preferredContentSizeCategory];

    NSNumber *cellHeight = cellHeightDictionary[userSize];
    [self.tableView setRowHeight:cellHeight.floatValue];
    [self.tableView reloadData];
}

```

Compile e execute o aplicativo. Se você alterar o tamanho do texto preferencial do tipo dinâmico e reiniciar o aplicativo mais uma vez, perceberá que a altura das linhas da visão de tabela refletirá o tamanho do texto selecionado pelo usuário. (Se você não reiniciar o aplicativo a partir do Xcode, terá de ir até a **BNRDetailViewController** e então retornar para a **BNRItemsViewController**.)

Assim como você fez com a **BNRDetailViewController** anteriormente, você precisará fazer com que a **BNRItemsViewController** registre-se como um observador para a **UIContentSizeCategoryDidChangeNotification**.

No **BNRItemsViewController.m**, registre-se para a notificação no **init** e remova o controlador de visão como observador no **dealloc**. Por fim, implemente o retorno de notificação para chamar o método **updateTableViewForDynamicTypeSize** que você acabou de criar.

```

self.navigationItem.rightBarButtonItem = bbi;
self.navigationItem.leftBarButtonItem = [self editButtonItem];

NSNotificationCenter *nc = [NSNotificationCenter defaultCenter];
[nc addObserver:self
    selector:@selector(updateTableViewForDynamicTypeSize)
    name:UIContentSizeCategoryDidChangeNotification
    object:nil];
}

return self;
}

- (void)dealloc
{
    NSNotificationCenter *nc = [NSNotificationCenter defaultCenter];
    [nc removeObserver:self];
}

```

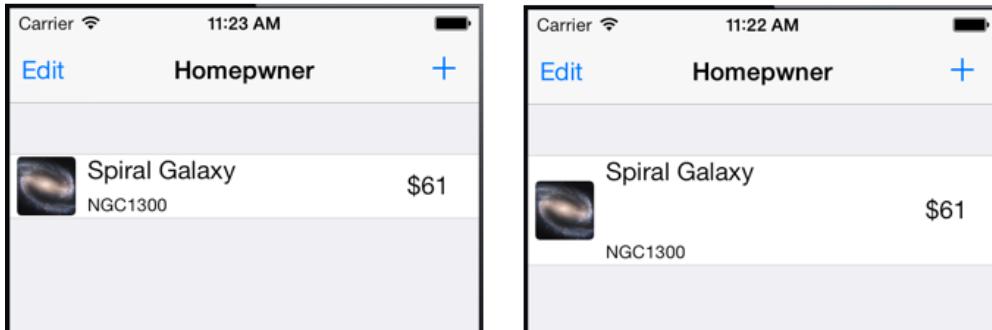
A altura da linha da visão de tabela será atualizada, assim que o tamanho do texto preferencial do tipo dinâmico for alterado. Compile e execute o aplicativo, e faça o teste.

## Atualização de BNRItemCell

Embora a altura das linhas se ajustem com base no tamanho de texto preferencial, o conteúdo da célula não é ajustado. Você adicionará o suporte Dynamic Type à **BNRItemCell** em breve. Primeiro, preste atenção em como

as subdivisões da célula se ajustam adequadamente com base na altura da célula. Por causa do uso cuidadoso do Auto Layout ao criar a `BNRItemCell`, a interface se ajusta perfeitamente (Figure 20.5). A `imageView` continua centralizada verticalmente na célula e fixada à esquerda, `nameLabel` é fixada na parte superior, `serialNumberLabel` é fixada na parte inferior, e `valueLabel` se mantém centralizada e fixada à direita.

Figure 20.5 Layout automático em ação



Tudo funcionou muito bem enquanto o texto estava com um tamanho fixo, porém, quando você atualizar a classe para usar o tipo dinâmico, precisará fazer algumas alterações. Vamos começar implementando o código Dynamic Type para atualizar os rótulos, que seguirão atentamente o que você fez com `BNRDetailViewController` e `BNRItemsViewController`.

Abra o `BNRItemCell.m` e faça as seguintes alterações:

```
- (void)updateInterfaceForDynamicTypeSize
{
    UIFont *font = [UIFont preferredFontForTextStyle:UIFontTextStyleBody];
    self.nameLabel.font = font;
    self.serialNumberLabel.font = font;
    self.valueLabel.font = font;
}

- (void)awakeFromNib
{
    [self updateInterfaceForDynamicTypeSize];

    NSNotificationCenter *nc = [NSNotificationCenter defaultCenter];
    [nc addObserver:self
        selector:@selector(updateInterfaceForDynamicTypeSize)
        name:UIContentSizeCategoryDidChangeNotification
        object:nil];
}

- (void)dealloc
{
    NSNotificationCenter *nc = [NSNotificationCenter defaultCenter];
    [nc removeObserver:self];
}
```

A única informação nova aqui é o método `awakeFromNib`. Ele é chamado em um objeto após ser desarquivado de um arquivo NIB, e é um ótimo local para realizar qualquer trabalho adicional de interface do usuário que não pode ser feito dentro do arquivo XIB. Você pode fazer qualquer configuração adicional da célula que não poderia ser feita dentro do arquivo XIB nesse método. Compile e execute o aplicativo; o texto será atualizado para refletir o tamanho do texto preferencial do usuário.

Existem alguns problemas no Auto Layout que você precisa resolver.

Inicialmente, você fixou a altura de `nameLabel` e `serialNumberLabel`. Isso não funciona bem agora que o tamanho do texto pode ser alterado dinamicamente.

No `BNRItemCell.xib`, exclua as duas restrições de altura desses dois rótulos. Se alguma visão estiver colocada no lugar errado, abra o menu `Resolve Auto Layout Issues` e selecione `Update All Frames in Homepwner Item Cell`. Compile e execute o aplicativo. A altura dos rótulos se baseará no tamanho do texto preferencial.

A interface está fantástica, mas não seria melhor se o tamanho de `imageView` refletisse o tamanho da fonte preferencial? É compreensível que a imagem deva ser dimensionada de acordo com o tamanho do texto. Vamos fazer isso.

## Outlets de restrição

Para atualizar a posição ou o tamanho de uma visão, tanto em termos absolutos quanto em relação a outra visão, você deverá atualizar as restrições dessa visão. Isso é muito importante! Se você modificar `frame` (ou `bounds`), em vez das restrições, na próxima vez que a visão precisar ser disposta, ela será disposta com base nas restrições que possui. Em outras palavras, as alterações feitas na `frame` não serão mantidas.

Para alterar a largura e a altura da visão de imagem, as constantes nas respectivas restrições precisarão ser atualizadas no tempo de execução. Para isso, você precisará criar um outlet para as restrições vertical e horizontal. As restrições são objetos (`NSLayoutConstraint`), portanto, da mesma maneira que é possível criar outlets para visões, é possível fazer com as restrições.

No `BNRItemCell.m`, crie e conecte o outlet para essas duas restrições à extensão de classe. Quando terminar, a extensão de classe será semelhante à seguinte:

```
@interface BNRItemCell ()
```

```
@property (nonatomic, weak) IBOutlet NSLayoutConstraint *imageViewHeightConstraint;
@property (nonatomic, weak) IBOutlet NSLayoutConstraint *imageViewWidthConstraint;
```

```
@end
```

Com os outlets para as restrições de tamanho da `imageView` criados, agora é possível ajustar o tamanho da `imageView` de forma programática. No `BNRItemCell.m`, modifique `updateInterfaceForDynamicTypeSize` para obter o tamanho de texto preferencial atualmente selecionado, e use isso para ajustar o tamanho da `imageView`.

```
- (void)updateInterfaceForDynamicTypeSize
{
    UIFont *font = [UIFont preferredFontForTextStyle:UIFontTextStyleBody];
    self.nameLabel.font = font;
    self.serialNumberLabel.font = font;
    self.valueLabel.font = font;

    static NSDictionary *imageSizeDictionary;

    if (!imageSizeDictionary) {
        imageSizeDictionary = @{
            UIFontContentSizeCategoryExtraSmall : @40,
            UIFontContentSizeCategorySmall : @40,
            UIFontContentSizeCategoryMedium : @40,
            UIFontContentSizeCategoryLarge : @40,
            UIFontContentSizeCategoryExtraLarge : @45,
            UIFontContentSizeCategoryExtraExtraLarge : @55,
            UIFontContentSizeCategoryExtraExtraExtraLarge : @65
        };
    }

    NSString *userSize =
        [[UIApplication sharedApplication] preferredContentSizeCategory];

    NSNumber *imageSize = imageSizeDictionary[userSize];
    self.imageViewHeightConstraint.constant = imageSize.floatValue;
    self.imageViewWidthConstraint.constant = imageSize.floatValue;
}
```

Compile e execute o aplicativo, e brinque com os tamanhos de texto do tipo dinâmico. A `imageView` ajusta seu tamanho de forma apropriada. Observe também que, como você fixou a borda esquerda de `nameLabel` e `serialNumberLabel` na borda direita da `imageView`, a interface é dimensionada muito bem de acordo com as alterações do tamanho de texto preferencial. Se, em vez disso, esses dois rótulos estivessem fixados à borda esquerda da supervisão, a `imageView` teria sobreposto os rótulos.

A interface está ótima, mas vamos fazer uma alteração final.

## Restrições de placeholder

Atualmente, você está atualizando as restrições de largura e altura para a `imageView`. Não há problema nenhum nisso, mas você pode fazer melhor. Em vez de atualizar ambas as restrições, você adicionará uma outra restrição à `imageView` que restringirá a largura e a altura da `imageView` para que sejam iguais.

Você não pode criar essa restrição no Interface Builder, por isso, retorne para o `BNRItemCell.m` e crie essa restrição de forma programática no `awakeFromNib`.

```
- (void)awakeFromNib
{
    [self updateInterfaceForDynamicTypeSize];

    NSNotificationCenter *nc = [NSNotificationCenter defaultCenter];
    [nc addObserver:self
        selector:@selector(updateInterfaceForDynamicTypeSize)
        name:UIContentSizeCategoryDidChangeNotification
        object:nil];

    NSLayoutConstraint *constraint =
        [NSLayoutConstraint constraintWithItem:self.thumbnailView
            attribute:NSLayoutAttributeHeight
            relatedBy:NSLayoutRelationEqual
            toItem:self.thumbnailView
            attribute:NSLayoutAttributeWidth
            multiplier:1
            constant:0];
    [self.thumbnailView addConstraint:constraint];
}
```

Agora, remova a propriedade de `imageViewWidthConstraint` e o código correspondente.

```
@interface BNRItemCell ()

@property (nonatomic, weak) IBOutlet NSLayoutConstraint *imageViewHeightConstraint;
@property (nonatomic, weak) IBOutlet NSLayoutConstraint *imageViewWidthConstraint;

@end

@implementation

- (void)updateInterfaceForDynamicTypeSize
{
    // Other code here

    NSNumber *imageSize = imageSizeDictionary[userSize];
    self.imageViewHeightConstraint.constant = imageSize.floatValue;
    self.imageViewWidthConstraint.constant = imageSize.floatValue;
}
```

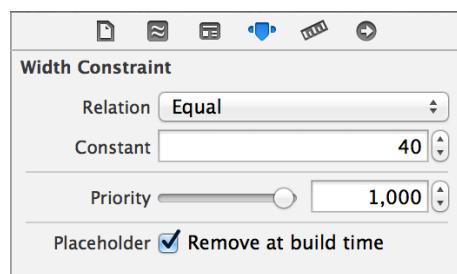
Abra o `BNRItemCell.xib` e certifique-se de que o outlet de `imageViewWidthConstraint` tenha sido removido, pois o outlet não existe mais.

Há uma alteração final que precisa ser feita. Existem agora duas restrições que afetam a largura da `imageView`: a restrição programática que você acabou de criar e a restrição explícita de largura no arquivo XIB. Serão criadas restrições insatisfatórias (conflitantes) caso as duas restrições não entrem em acordo em relação ao tamanho.

Para corrigir isso, é possível excluir a restrição explícita de largura da `imageView`. Isso resolverá, porém o Interface Builder poderá alertá-lo quanto a visões posicionadas incorretamente ou layout ambíguo. Em vez disso, você pode transformar a restrição de largura em *restrição de placeholder*. Restrições de placeholder, como o nome já diz, são apenas temporárias e são removidas no momento da compilação, portanto não existirão quando o aplicativo estiver sendo executado.

No `BNRItemCell.xib`, selecione a restrição de largura na `imageView` e abra o inspetor de atributos. Marque a caixa Placeholder que diz Remove at build time (Figure 20.6). Compile e execute o aplicativo. Tudo funcionará como antes. O Homeowner agora é dimensionado apropriadamente de acordo com o tamanho de texto preferencial do usuário.

Figure 20.6 Restrições de placeholder





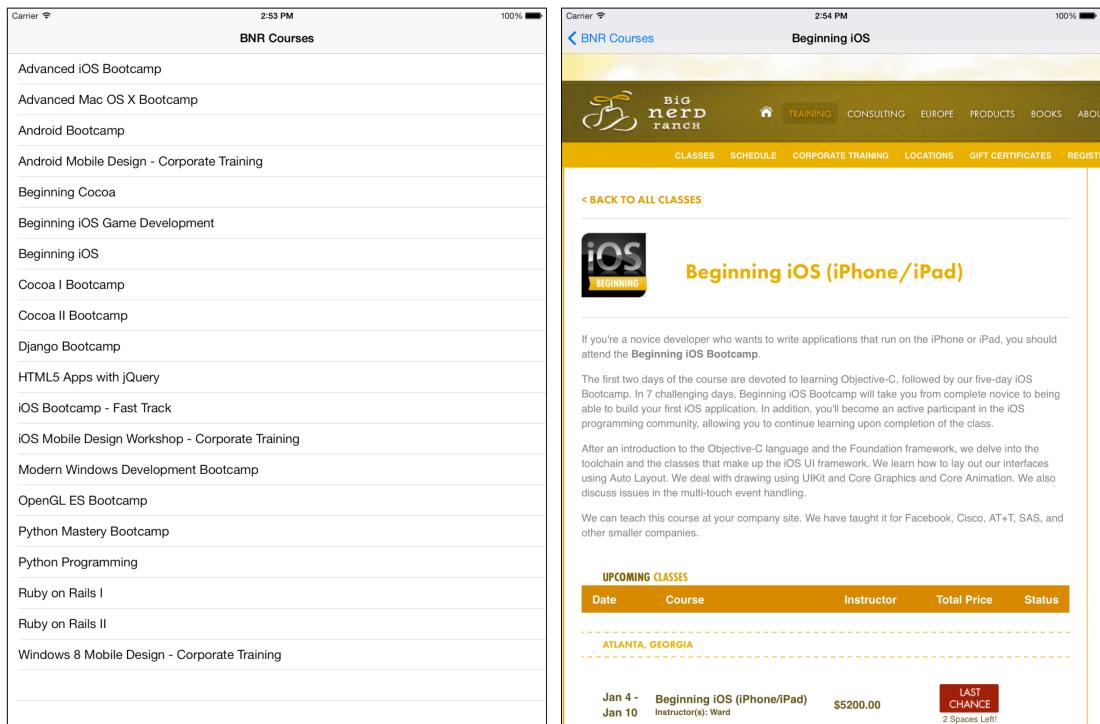
# 21

## Serviços web e UIWebView

Nos próximos dois capítulos, você irá dar outra pausa no Homepwner para trabalhar com serviços web e controladores de visão dividida.

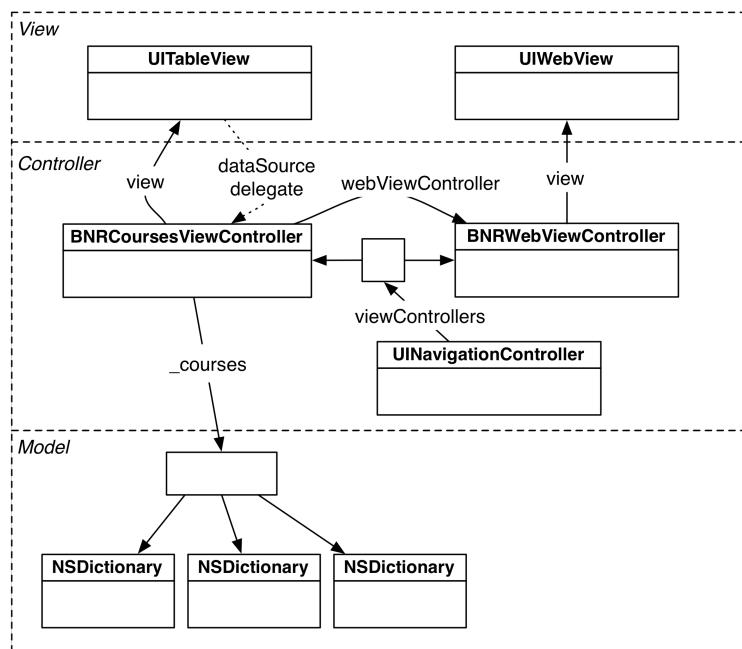
Neste capítulo, você estabelecerá a base para um aplicativo chamado Nerdfeed que lê uma lista dos cursos que a Big Nerd Ranch oferece. Cada curso será listado em uma visão de tabela, e selecionar um curso abrirá a página da web do curso em questão. A Figure 21.1 mostra o Nerdfeed ao final deste capítulo.

Figure 21.1 Nerdfeed



O trabalho é dividido em duas partes. A primeira é se conectar e coletar dados de um serviço web, e usar esses dados para criar objetos de modelo. A segunda parte é usar a classe **UIWebView** para exibir conteúdo web. A Figure 21.2 mostra o diagrama de objetos do Nerdfeed.

Figure 21.2 Diagrama de objetos do Nerdfeed



## Serviços web

Seu navegador web usa o protocolo HTTP para se comunicar com um servidor web. No tipo mais simples de interação, o navegador envia uma solicitação ao servidor especificando um URL. O servidor responde enviando de volta a página solicitada (geralmente HTML e imagens), a qual o navegador formata e exibe.

Em interações mais complexas, as solicitações do navegador incluem outros parâmetros, como dados de formulários. O servidor processa esses parâmetros e retorna uma página web customizada ou dinâmica.

Os navegadores web são amplamente utilizados e já existem há bastante tempo. Portanto, as tecnologias que envolvem o HTTP são estáveis e bem-desenvolvidas: o tráfego HTTP passa ordenadamente pela maioria dos firewalls, os servidores web são bastante seguros e têm excelente desempenho, e as ferramentas de desenvolvimento de aplicativos web já estão muito fáceis de usar.

Você pode escrever um aplicativo cliente para iOS que aproveite a infraestrutura HTTP para conversar com um servidor habilitado para web. O lado servidor desse aplicativo é um *serviço web*. O seu aplicativo cliente e o serviço web podem trocar solicitações e respostas via HTTP.

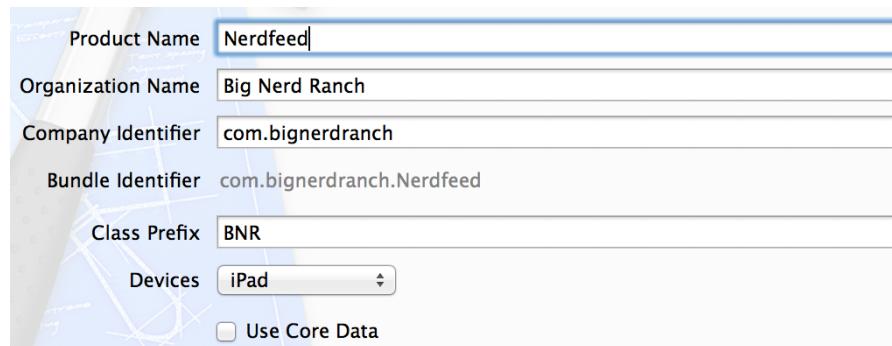
Como o protocolo HTTP não se importa com o tipo de dados que está transportando, essas trocas podem conter dados complexos. Esses dados estão normalmente nos formatos JSON (JavaScript Object Notation) ou XML. Se você controla o servidor web e o cliente também, você pode usar qualquer formato que desejar; caso contrário, você terá que construir seu aplicativo para usar o formato suportado pelo servidor.

Neste capítulo, você vai criar um aplicativo cliente que fará uma solicitação ao serviço web courses hospedado em <http://bookapi.bignerdranch.com>. Os dados que são retornados serão JSON que descrevem os cursos.

## Iniciando a construção do aplicativo Nerdfeed

Crie um novo Empty Application para a família de dispositivos iPad. Dê a esse aplicativo o nome de Nerdfeed, conforme mostrado na Figure 21.3. (Se você não tem um iPad no qual implementar, use o simulador de iPad.)

Figure 21.3 Criação de um aplicativo vazio para iPad



Vamos resolver a questão da interface do usuário básica antes de nos concentrarmos nos serviços web. Crie uma nova subclasse **NSObject** e nomeie-a como **BNRCoursesViewController**. No **BNRCoursesViewController.h**, altere a superclasse para **UITableViewViewController**.

```
@interface BNRCoursesViewController : NSObject
@interface BNRCoursesViewController : UITableViewViewController
```

No **BNRCoursesViewController.m**, escreva stubs para os métodos de fonte de dados necessários, para que você possa compilar e executar conforme avança neste exercício.

```
- (NSInteger)tableView:(UITableView *)tableView
 numberOfRowsInSection:(NSInteger)section
{
    return 0;
}

- (UITableViewCell *)tableView:(UITableView *)tableView
    cellForRowAtIndexPath:(NSIndexPath *)indexPath
{
    return nil;
}
```

No **BNRAppDelegate.m**, crie uma instância de **BNRCoursesViewController** e configure-a como o controlador de visão raiz de um controlador de navegação. Agora transforme o controlador de navegação no controlador de visão raiz da janela.

```
#import "BNRAppDelegate.h"
#import "BNRCoursesViewController.h"

@implementation BNRAppDelegate

- (BOOL)application:(UIApplication *)application
    didFinishLaunchingWithOptions:(NSDictionary *)launchOptions
{
    self.window = [[UIWindow alloc] initWithFrame:[[UIScreen mainScreen] bounds]];

    BNRCoursesViewController *cvc =
        [[BNRCoursesViewController alloc] initWithStyle:UITableViewStylePlain];

    UINavigationController *masterNav =
        [[UINavigationController alloc] initWithRootViewController:cvc];

    self.window.rootViewController = masterNav;

    self.window.backgroundColor = [UIColor whiteColor];
    [self.window makeKeyAndVisible];

    return YES;
}
```

Compile e execute o aplicativo. Você deve ver uma **UITableView** vazia e uma barra de navegação.

## NSURL, NSURLRequest, NSURLSession e NSURLSessionTask

O aplicativo Nerdfeed vai obter dados de um servidor web usando quatro classes úteis: **NSURL**, **NSURLRequest**, **NSURLSessionTask** e **NSURLSession** (Figure 21.4).

Figure 21.4 Relacionamento entre classes de serviços web



Cada uma dessas três classes tem um papel importante na comunicação com um servidor web:

- Uma instância de **NSURL** contém a localização de um aplicativo web no formato URL. Para muitos serviços web, o URL será composto pelo endereço de base, o aplicativo web com o qual você está se comunicando e quaisquer argumentos que estejam sendo passados.
- Uma instância de **NSURLRequest** armazena todos os dados necessários para se comunicar com um servidor web. Isso inclui um objeto **NSURL**, bem como uma política de cache, um limite de tempo para que o servidor web responda e dados adicionais passados por meio do protocolo HTTP. (A **NSMutableURLRequest** é a subclasse mutável da **NSURLRequest**.)
- Uma instância de **NSURLSessionTask** engloba o tempo de vida de uma única solicitação. Ela rastreia o estado da solicitação e tem métodos para cancelar, suspender e retomar a solicitação. Tarefas sempre serão subclasses de **NSURLSessionTask**, especificamente, **NSURLSessionDataTask**, **NSURLSessionUploadTask** ou **NSURLSessionDownloadTask**.
- Uma instância de **NSURLSession** atua como uma fábrica de tarefas de transferência de dados. É um recipiente configurável que pode estabelecer propriedades comuns nas tarefas que ela cria. Alguns exemplos incluem campos de cabeçalho que todas as solicitações devem ter ou se solicitações funcionam com conexões de celular. **NSURLSession** também tem um abrangente modelo de delegate que pode fornecer informações sobre o status de uma determinada tarefa e resolver desafios de autenticação, por exemplo.

## Formatação de URLs e solicitações

A formatação de uma solicitação de serviço web varia dependendo de quem implementa o serviço web; não há regras rígidas quando se trata de serviços web. Você terá que encontrar a documentação do serviço web para saber como formatar uma solicitação. Contanto que um aplicativo cliente envie ao servidor o que ele deseja obter, você terá um intercâmbio funcionando.

O serviço web dos cursos da Big Nerd Ranch precisa de um URL com o seguinte formato:

`http://bookapi.bignerdranch.com/courses.json`

Você pode ver que o URL de base é `bookapi.bignerdranch.com` e que o aplicativo web fica localizado em `courses` (cursos) no arquivo de sistemas do servidor em questão, seguido pelo formato de dados (`json`) no qual você espera a resposta.

Solicitações de serviço web vêm em todos os tipos de formatos, dependendo do que o criador do serviço web em questão estiver tentando fazer. O serviço web de `courses`, onde cada informação de que o servidor web precisa para honrar uma solicitação é dividida em argumentos fornecidos como componentes de caminho, é um tanto quanto comum. Essa chamada de serviço web em particular diz: “retorne todos os cursos em um formato `json`”.

Outro formato de serviço web comum é semelhante ao seguinte:

`http://baseURL.com/serviceName?argumentX=valueX&argumentY=valueY`

Então, por exemplo, você pode imaginar que poderia especificar o mês e o ano para os quais deseja uma lista de cursos com um URL como o seguinte:

```
http://bookapi.bignerdranch.com/courses.json?year=2014&month=11
```

Às vezes, é preciso que você faça uma string ser “segura para URL”. Por exemplo, caracteres de espaço e aspas não são permitidos em URLs; eles devem ser substituídos por sequências de escape. Isso é feito da seguinte forma:

```
NSString *search = @"Play some \"Abba\"";  
NSString *escaped =  
    [search stringByAddingPercentEscapesUsingEncoding:NSUTF8StringEncoding];  
  
// escaped is now "Play%20some%20%22Abba%22"
```

Se você precisar “desescapar” uma string com escape percentual, **NSString** tem o método:

```
- (NSString *)stringByRemovingPercentEncoding;
```

Quando a solicitação do feed de cursos da Big Nerd Ranch for processado, o servidor retornará dados JSON que contêm a lista de cursos. A **BNRCoursesViewController**, que fez a solicitação, preencherá sua visão de tabela com os títulos desses cursos.

## Trabalhando com NSURLConnection

**NSURLSession** se refere tanto a uma classe específica quanto a um nome de uma coleção de classes que formam uma API para solicitações de rede. Para diferenciar as duas definições, o livro se referirá à classe como simplesmente **NSURLSession**, ou uma instância da mesma, e à API como a **API da NSURLSession**.

No **BNRCoursesViewController.m**, adicione uma propriedade à extensão de classe para se fixar a uma instância de **NSURLSession**.

```
@interface BNRCoursesViewController ()  
  
@property (nonatomic) NSURLSession *session;  
  
@end
```

Em seguida, sobrescreva **initWithStyle:** para criar o objeto **NSURLSession**.

```
- (instancetype)initWithStyle:(UITableViewStyle)style  
{  
    self = [super initWithStyle:style];  
    if (self) {  
        self.navigationItem.title = @"BNR Courses";  
  
        NSURLSessionConfiguration *config =  
            [NSURLSessionConfiguration defaultSessionConfiguration];  
        _session = [NSURLSession sessionWithConfiguration:config  
                                         delegate:nil  
                                         delegateQueue:nil];  
    }  
    return self;  
}
```

A **NSURLSession** é criada com uma configuração, um delegate e uma fila de delegate. Os padrões para esses argumentos são o que você quer para esse aplicativo. Você recebe uma configuração padrão e a passa para o primeiro argumento. Para o segundo e terceiro argumentos, você simplesmente passa `nil` para obter os padrões.

No **BNRCoursesViewController.m**, implemente o método **fetchFeed** para criar uma **NSURLRequest** que se conecte a `bookapi.bignerdranch.com` e solicite a lista de cursos. Depois, crie a **NSURLSession** para criar uma **NSURLSessionDataTask** que transfira essa solicitação ao servidor.

```

- (void)fetchFeed
{
    NSString *requestString = @"http://bookapi.bignerdranch.com/courses.json";
    NSURL *url = [NSURL URLWithString:requestString];
    NSURLRequest *req = [NSURLRequest requestWithURL:url];

    NSURLSessionDataTask *dataTask =
        [self.session dataTaskWithRequest:req
            completionHandler:
            ^(NSData *data, NSURLResponse *response, NSError *error) {
                NSString *json = [[NSString alloc] initWithData:data
                                                encoding:NSUTF8StringEncoding];
                NSLog(@"%@", json);
            }];
    [dataTask resume];
}

```

Criar a **NSURLRequest** é relativamente simples: crie uma instância da **NSURL** e instancie um objeto de solicitação com ela.

A finalidade da **NSURLSession** é um pouco mais obscura. O trabalho da **NSURLSession** é criar tarefas de natureza semelhante. Por exemplo, se seu aplicativo tiver um conjunto de solicitações, todas exigindo os mesmos campos de cabeçalho, você poderia configurar a **NSURLSession** com esses campos de cabeçalho adicionais. De forma semelhante, se um conjunto de solicitações não se conectar em redes de celular, uma **NSURLSession** poderia ser configurada para não permitir o acesso por celulares. Esses comportamentos e atributos compartilhados são, então, configurados nas tarefas que a sessão cria.

Um projeto pode ter múltiplas instâncias de **NSURLSession**, mas, já que o Nerdfeed só inicia uma única solicitação simples, ele usará o **sharedSession**. O **sharedSession** é definido com uma configuração padrão.

O objeto da sessão já pode ser usado para criar tarefas. Ao dar à seção uma solicitação e um bloco de conclusão para chamar quando a solicitação se encerrar, ela retornará uma instância de **NSURLSessionTask**. Como o Nerdfeed está solicitando dados de um serviço web, o tipo de tarefa será **NSURLSessionDataTask**. Tarefas sempre são criadas no estado **suspended**; por isso, chamar **resume** na tarefa iniciará a solicitação do serviço web. Por enquanto, o bloco de conclusão imprimirá apenas os dados JSON retornados a partir da solicitação.

Inicie o intercâmbio assim que a **BNRCoursesViewController** estiver criada. No **BNRCoursesViewController.m**, atualize o **initWithStyle**:

```

- (instancetype)initWithStyle:(UITableViewStyle)style
{
    self = [super initWithStyle:style];
    if (self) {
        self.navigationItem.title = @"BNR Courses";

        NSURLSessionConfiguration *config =
            [NSURLSessionConfiguration defaultSessionConfiguration];
        _session = [NSURLSession sessionWithConfiguration:config
                                                delegate:nil
                                               delegateQueue:nil];
        [self fetchFeed];
    }
    return self;
}

```

Compile e execute o aplicativo. Uma representação de string dos dados JSON que retornam do serviço web será impressa para o console. (Se você não vir nada sendo impresso para o console, certifique-se de ter digitado corretamente o URL.)

## Dados JSON

Dados JSON, especialmente quando são condensados como estão no seu console, podem parecer complexos. No entanto, na realidade, é uma sintaxe muito simples. JSON pode conter os objetos mais básicos que usamos para representar objetos de modelo: arrays, dicionários, strings e números. Um dicionário contém um ou mais pares chave-valor, onde a chave é uma string, e o valor pode ser outro dicionário, string, número ou array.

Um array pode consistir em strings, números, dicionários e outros arrays. Assim, um documento JSON é um conjunto aninhado desses tipos de valores. Veja um exemplo de alguns JSON bem simples:

```
{
    "name" : "Christian",
    "friends" : ["Aaron", "Mikey"],
    "job" : {
        "company" : "Big Nerd Ranch",
        "title" : "Senior Nerd"
    }
}
```

Um dicionário JSON é delimitado por chaves ({ e }). Dentro de chaves, ficam os pares chave-valor que pertencem ao dicionário em questão. O início desse documento JSON é uma chave aberta, o que significa que o objeto de nível mais alto desse documento é um dicionário JSON. Esse dicionário contém três pares chave-valor (nome, amigos e emprego).

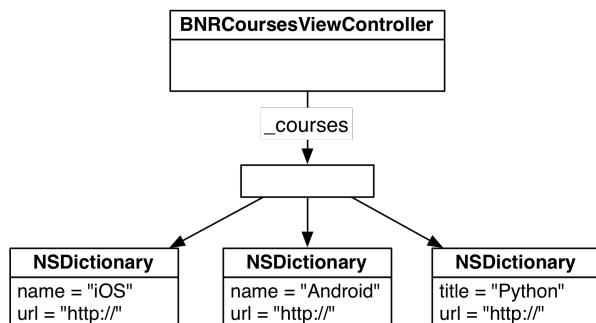
Uma string é representada pelo uso de texto dentro de aspas. Strings são usadas como as chaves dentro de um dicionário e podem ser usadas como valores também. Assim, o valor da chave nome no dicionário de nível mais alto é a string Christian.

Arrays são representados por colchetes ([ e ]). Um array pode conter qualquer outra informação JSON. Neste caso, a chave amigos contém um array de strings (Aaron e Mikey).

Um dicionário pode conter outros dicionários; a chave final do dicionário de nível mais alto, emprego, é associada a um dicionário que tenha dois pares chave-valor (empresa e cargo).

O Nerdfeed analisará as informações úteis a partir dos dados JSON e armazenará isso em sua propriedade courses.

Figure 21.5 Gráfico de objetos de modelo



## Análise de dados JSON

A Apple tem uma classe embutida para analisar dados JSON, **NSJSONSerialization**. Você pode entregar a essa classe um monte de dados JSON, e ela criará instâncias de **NSDictionary** para cada objeto JSON, **NSArray** para cada array JSON, **NSString** para cada string JSON e **NSNumber** para cada número JSON.

No BNRCoursesViewController.m, modifique o handler de conclusão da **NSURLSessionDataTask** para usar a classe **NSJSONSerialization** para converter os dados JSON brutos em objetos de fundação básicos.

```

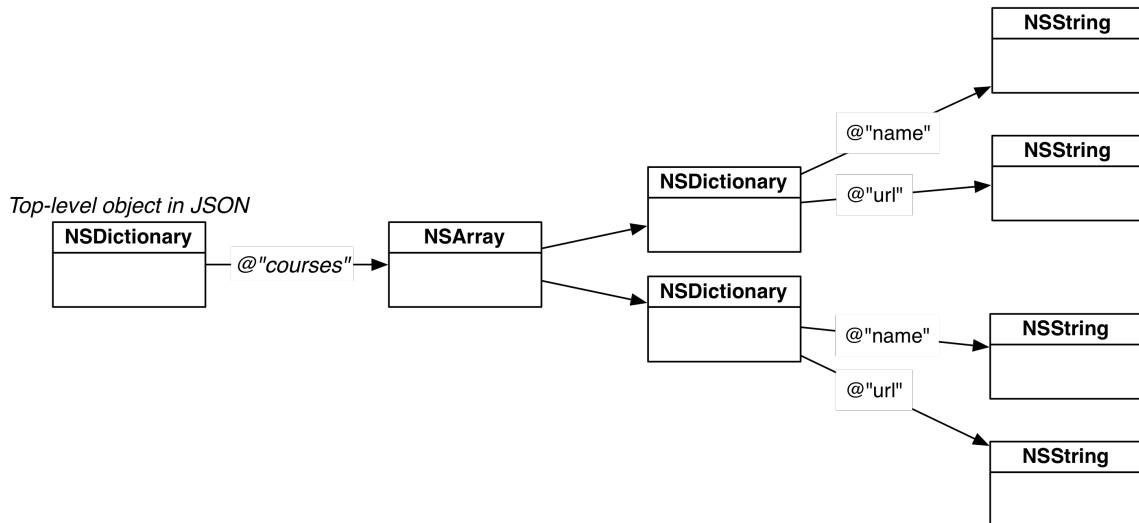
^(NSData *data, NSURLResponse *response, NSError *error) {
    NSString *json = [[NSString alloc] initWithData:data
                                              encoding:NSUTF8StringEncoding];
    NSLog(@"%@", json);

    NSDictionary *jsonObject = [NSJSONSerialization JSONObjectWithData:data
                                                               options:0
                                                               error:nil];
    NSLog(@"%@", jsonObject);
}
  
```

Compile e execute; depois, verifique o console. Você verá os dados JSON, só que, agora, eles estarão formatados de maneira ligeiramente diferente, porque a **NSLog** faz um bom trabalho formatando dicionários e arrays. A **jsonObject** é uma instância de **NSDictionary** e tem uma chave **NSString** com um valor associado do tipo **NSArray**.

Quando a **NSURLSessionDataTask** concluir, você usará **NSJSONSerialization** para converter os dados JSON em um **NSDictionary**. A Figure 21.6 mostra como os dados serão estruturados.

Figure 21.6 Objetos JSON



No **BNRCoursesViewController.m**, adicione uma nova propriedade à extensão de classe para se fixar ao array em questão, que é um array de objetos **NSDictionary** que descrevem cada curso.

```

@interface BNRCoursesViewController ()

@property (nonatomic) NSURLSession *session;
@property (nonatomic, copy) NSArray *courses;

@end
  
```

Então, no mesmo arquivo, mude a implementação do handler de conclusão **NSURLSessionDataTask**:

```

^(NSData *data, NSURLResponse *response, NSError *error) {
    NSDictionary *jsonObject = [NSJSONSerialization JSONObjectWithData:data
                                                               options:0
                                                               error:nil];
    NSLog(@"%@", jsonObject);
    self.courses = jsonObject[@"courses"];
    NSLog(@"%@", self.courses);
}
  
```

Agora, ainda no **BNRCoursesViewController.m**, atualize os métodos de fonte de dados para que cada um dos títulos de curso seja mostrado na tabela. Também será bom sobrescrever **viewDidLoad** para registrar a classe de célula da visão de tabela.

```

- (void)viewDidLoad
{
    [super viewDidLoad];

    [self.tableView registerClass:[UITableViewCell class]
        forCellReuseIdentifier:@"UITableViewCell"];
}

- (NSInteger)tableView:(UITableView *)tableView
 numberOfRowsInSection:(NSInteger)section
{
    return 0;
    return [self.courses count];
}

- (UITableViewCell *)tableView:(UITableView *)tableView
    cellForRowAtIndexPath:(NSIndexPath *)indexPath
{
    return nil;
    UITableViewCell *cell =
        [tableView dequeueReusableCellWithIdentifier:@"UITableViewCell"
            forIndexPath:indexPath];

    NSDictionary *course = self.courses[indexPath.row];
    cell.textLabel.text = course[@"title"];

    return cell;
}

```

## O thread principal

Dispositivos modernos com iOS têm processadores com vários núcleos que permitem aos dispositivos executar múltiplos pedaços (chunks) de código de forma simultânea. De forma bastante adequada, isso é chamado de *simultaneidade*, e cada pedaço de código é executado em um *thread* separado. Até agora neste livro, todos os nossos códigos foram executados no *thread principal*. O thread principal é, por vezes, chamado de thread de IU (interface do usuário), pois qualquer código que modifique a IU precisa ser executado no thread principal.

Quando o serviço web é concluído, você precisa carregar novamente os dados de visão de tabela. Como padrão, `NSURLSessionDataTask` executa o handler de conclusão em um thread em segundo plano. Você precisa de uma maneira de forçar o código a ser executado no thread principal para carregar novamente a visão de tabela, e pode fazer isso facilmente usando a função `dispatch_async`.

No `BNRCoursesViewController.m`, atualize o handler de conclusão para carregar novamente os dados de visão de tabela no thread principal.

```

^(NSData *data, NSURLResponse *response, NSError *error) {
    NSDictionary *jsonObject = [NSJSONSerialization JSONObjectWithData:data
        options:0
        error:nil];
    self.courses = jsonObject[@"courses"];
    NSLog(@"%@", self.courses);

    dispatch_async(dispatch_get_main_queue(), ^{
        [self.tableView reloadData];
    });
}

```

Compile e execute o aplicativo. Depois de o serviço web ser concluído, você deverá ver uma lista com os cursos da Big Nerd Ranch.

## UIWebView

Além de seu título, o dicionário de cada curso também mantém uma string de URL que aponta para sua página na web. Seria bom se o Nerdfeed pudesse abrir o Safari e navegar até esse URL. Seria melhor ainda se o Nerdfeed pudesse abrir essa página web sem precisar sair do Nerdfeed para abrir o Safari. Uma boa notícia: ele pode usar a classe `UIWebView`.

As instâncias da **UIWebView** renderizam conteúdo web. Na verdade, o aplicativo Safari no seu dispositivo usa uma **UIWebView** para renderizar seu conteúdo web. Nesta parte do capítulo, você vai criar um controlador de visão cuja visão é uma instância de **UIWebView**. Quando um dos itens é selecionado na visão de tabela de cursos, você adiciona o controlador da visão web à pilha de navegação e faz com que ele carregue o URL armazenado em **NSDictionary**.

Crie uma nova subclasse de **NSObject** e nomeia-a como **BNRWebViewController**. No **BNRWebViewController.h**, adicione uma propriedade e altere a superclasse para **UIViewController**:

```
@interface BNRWebViewController : NSObject
@interface BNRWebViewController : UIViewController

@property (nonatomic) NSURL *URL;

@end
```

No **BNRWebViewController.m**, escreva a implementação a seguir.

```
@implementation BNRWebViewController

- (void)loadView
{
    UIWebView *webView = [[UIWebView alloc] init];
    webView.scalesPageToFit = YES;
    self.view = webView;
}

- (void)setURL:(NSURL *)URL
{
    _URL = URL;
    if (_URL) {
        NSURLRequest *req = [NSURLRequest requestWithURL:_URL];
        [(UIWebView *)self.view loadRequest:req];
    }
}
@end
```

No **BNRCoursesViewController.h**, adicione uma nova propriedade para se fixar a uma instância de **BNRWebViewController**.

```
@class BNRWebViewController;

@interface BNRCoursesViewController : UITableViewController

@property (nonatomic) BNRWebViewController *webViewController;

@end
```

No **BNRAppDelegate.m**, importe o cabeçalho para **BNRWebViewController**, crie uma instância de **BNRWebViewController** e defina-a como **BNRWebViewController** da **BNRCoursesViewController**.

```

#import "BNRWebViewController.h"

@interface BNRCoursesViewController : UIViewController
@property (nonatomic) NSURLSession *session;
@property (nonatomic, copy) NSArray *courses;
@end

@implementation BNRAppDelegate
- (BOOL)application:(UIApplication *)application
didFinishLaunchingWithOptions:(NSDictionary *)launchOptions
{
    self.window = [[UIWindow alloc] initWithFrame:[[UIScreen mainScreen] bounds]];
    BNRCoursesViewController *cvc =
        [[BNRCoursesViewController alloc] initWithStyle:UITableViewStylePlain];
    UINavigationController *masterNav =
        [[UINavigationController alloc] initWithRootViewController:cvc];
    BNRWebViewController *wvc = [[BNRWebViewController alloc] init];
    lvc.webViewController = wvc;
    self.window.rootViewController = masterNav;
    self.window.backgroundColor = [UIColor whiteColor];
    [self.window makeKeyAndVisible];
    return YES;
}

```

(Observe que você está instanciando a **BNRWebViewController** no delegate do aplicativo como preparação para o próximo capítulo, onde você usará um **UISplitViewController** para exibir controladores de visão no iPad.)

No **BNRCoursesViewController.m**, importe os arquivos de cabeçalho para **BNRWebViewController** e, em seguida, implemente **tableView:didSelectRowAtIndexPath:** para configurar e adicionar **webViewController** à pilha de navegação quando uma linha for tocada.

```

#import "BNRWebViewController.h"

@implementation BNRCoursesViewController
- (void)tableView:(UITableView *)tableView
didSelectRowAtIndexPath:(NSIndexPath *)indexPath
{
    NSDictionary *course = self.courses[indexPath.row];
    NSURL *URL = [NSURL URLWithString:course[@"url"]];

    self.webViewController.title = course[@"title"];
    self.webViewController.URL = URL;
    [self.navigationController pushViewController:self.webViewController
                                         animated:YES];
}

```

Compile e execute o aplicativo. Você deve poder selecionar um dos cursos, e isso deverá levar você a um novo controlador de visão que exibirá a página web do curso em questão.

## Credenciais

Quando você tentar acessar um serviço web, às vezes, ele responderá com um *desafio de autenticação*, que significa “Afinal, quem é você?”. Você então precisa enviar um nome de usuário e uma senha (uma *credencial*) para que o servidor envie sua resposta autêntica.

Quando o desafio é recebido, o delegate de **NSURLSession** recebe o pedido de autenticar esse desafio, e o delegate responderá fornecendo um nome de usuário e senha.

Abra **BNRCoursesViewController.m** e atualize **fetchFeed** para ativar um serviço web seguro de cursos da Big Nerd Ranch. (Não se esqueça de usar **https**, em vez de **http**.)

```

- (void)fetchFeed
{
    NSString *requestString = @"http://bookapi.bignerdranch.com/courses.json";
    NSString *requestString = @"https://bookapi.bignerdranch.com/private/courses.json";

    NSURL *url = [NSURL URLWithString:requestString];
    NSURLRequest *req = [NSURLRequest requestWithURL:url];

```

Agora, a **NSURLSession** precisa que seu delegate ataque a criação. Atualize **initWithStyle:** para configurar o delegate da sessão.

```

- (instancetype)initWithStyle:(UITableViewStyle)style
{
    self = [super initWithStyle:style];
    if (self) {
        self.navigationItem.title = @"BNR Courses";

        NSURLSessionConfiguration *config =
            [NSURLSessionConfiguration defaultSessionConfiguration];
        _session = [NSURLSession sessionWithConfiguration:config
                                                delegate:nil
                                               delegateQueue:nil];
        _session = [NSURLSession sessionWithConfiguration:config
                                                delegate:self
                                               delegateQueue:nil];

        [self fetchFeed];
    }
    return self;
}

```

Em seguida, atualize a extensão de classe no **BNRCoursesViewController.m** para conformidade com o protocolo **NSURLSessionDataDelegate**.

```

@interface BNRCoursesViewController () <NSURLSessionDataDelegate>

@property (nonatomic) NSURLSession *session;
@property (nonatomic, copy) NSArray *courses;

@end

```

Compile e execute o aplicativo. O serviço web será concluído com um erro (acesso não autorizado), e nenhum dado será retornado. Por causa disso, **BNRCoursesViewController** não terá um array de cursos para exibir, e, assim, a visão de tabela permanecerá vazia.

Para autorizar essa solicitação, você precisará implementar o método do delegate do desafio de autenticação. Esse método fornecerá um bloco que você poderá chamar, passando as credenciais como um argumento.

No **BNRCoursesViewController.m** implemente o método **NSURLSessionDataDelegate** para lidar com o desafio de autenticação.

```

- (void)URLSession:(NSURLSession *)session task:(NSURLSessionTask *)task
didReceiveChallenge:(NSURLAuthenticationChallenge *)challenge
completionHandler:
{
    (void (^)(NSURLSessionAuthChallengeDisposition, NSURLCredential *))completionHandler
    {
        NSURLCredential *cred =
            [NSURLCredential credentialWithUser:@"BigNerdRanch"
                                         password:@"AchieveNerdvana"
                                         persistence:NSURLConnectionPersistenceForSession];
        completionHandler(NSURLSessionAuthChallengeUseCredential, cred);
    }
}

```

O handler de conclusão aceita dois argumentos. O primeiro argumento é o tipo de credenciais que você fornecerá. Como você vai fornecer um nome de usuário e senha, o tipo de autenticação é **NSURLSessionAuthChallengeUseCredential**. O segundo argumento são as próprias credenciais, uma instância de **NSURLCredential**, que é criada com o nome de usuário, senha e uma enumeração especificando durante quanto tempo essas credenciais deverão ser válidas.

Compile e execute o aplicativo, e ele se comportará como antes de você ter alterado o serviço web seguro, mas, agora, ele está buscando os cursos de forma segura por SSL.

## Desafio de prata: mais UIWebView

Uma **UIWebView** mantém seu próprio histórico. Você pode enviar as mensagens **goBack** e **goForward** a uma visão web e ela navegará por esse histórico. Crie uma instância de **UIToolbar** e adicione-a à hierarquia de visões de **BNRWebViewController**. Essa barra de ferramentas deve ter botões de avançar e voltar que permitirão que a visão web navegue por seu histórico. Bônus: use as duas outras propriedades da **UIWebView** para habilitar e desabilitar os itens da barra de ferramentas.

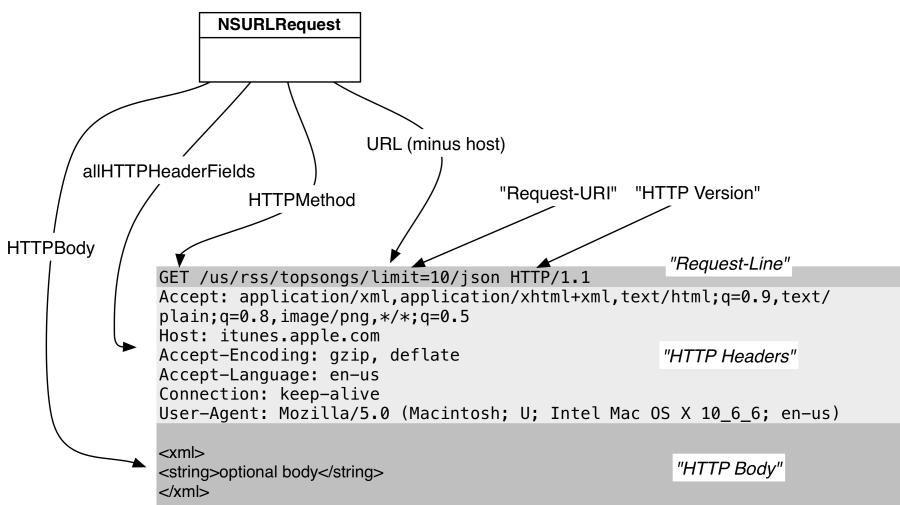
## Desafio de ouro: próximos cursos

Além de fornecer informações gerais de cursos, o serviço web cursos também retorna informações sobre os próximos cursos, como data e instrutor. Crie uma nova subclasse **UITableViewCell** que exiba o título do curso e a próxima vez em que o curso será oferecido. (Observação: nem todos os cursos têm uma próxima ocasião.)

## Para os mais curiosos: o corpo da solicitação

Quando **NSURLSessionTask** fala com um servidor web, ela usa o protocolo HTTP. Esse protocolo diz que todos os dados que você envia ou recebe devem seguir as especificações do HTTP. Os dados efetivamente transferidos ao servidor neste capítulo são mostrados na Figure 21.7.

Figure 21.7 Formato de solicitação HTTP



**NSURLRequest** tem uma série de métodos que permitem que você especifique uma parte da solicitação e depois formate corretamente para você.

Qualquer solicitação de serviço tem três partes: uma linha de solicitação, os cabeçalhos HTTP e o corpo HTTP, que é opcional. A linha de solicitação (que a Apple chama de linha de status) é a primeira linha da solicitação e diz ao servidor o que o cliente está tentando fazer. Nessa solicitação, o cliente está tentando dar um GET no recurso em `/courses.json`. (Ele também especifica a versão da especificação HTTP na qual os dados se encontram.)

O comando GET é um método HTTP. Embora exista uma série de métodos HTTP suportados, você verá mais comumente o GET e o POST. O padrão da **NSURLRequest**, GET, indica que o cliente quer alguma coisa *do* servidor. O que ele quer é conhecido como Request-URI (`/courses.json`).

No início da Internet, o Request-URI era o caminho de um arquivo no servidor. Por exemplo, a solicitação `http://www.website.com/index.html` retornava o arquivo `index.html` e seu navegador mostrava esse arquivo em uma janela. Hoje, ainda usamos o Request-URI para especificar um serviço que o servidor implementa. Por exemplo, neste capítulo, você acessou o serviço `cursos`, forneceu parâmetros a ele e recebeu como resposta um

documento JSON. Você ainda faz o GET de alguma coisa, mas o servidor é mais inteligente ao interpretar o que você está solicitando.

Além de obter coisas do servidor, você pode também enviar informações a ele. Por exemplo, muitos servidores web permitem que você faça upload de fotos. Um aplicativo cliente passaria os dados da imagem ao servidor por meio de uma solicitação de serviço. Nessa situação, você usa o método HTTP POST, que indica ao servidor que você está incluindo o corpo HTTP opcional. O corpo de uma solicitação são os dados que você pode incluir com a solicitação – geralmente dados codificados em XML, JSON ou Base-64.

Quando a solicitação tem um corpo, ela também precisa ter um cabeçalho Content-Length. De forma bastante conveniente, a **NSURLRequest** calcula o tamanho do corpo e adiciona esse cabeçalho para você.

```
NSURL *someURL = [NSURL URLWithString:@"http://www.photos.com/upload"];
UIImage *image = [self profilePicture];
NSData *data = UIImagePNGRepresentation(image);

NSMutableURLRequest *req =
    [NSMutableURLRequest requestWithURL:someURL
        cachePolicy:NSURLRequestReloadIgnoringCacheData
        timeoutInterval:90];

// This adds the HTTP body data and automatically sets the Content-Length header
req.HTTPBody = data;

// This changes the HTTP Method in the request-line
req.HTTPMethod = @"POST";

// If you wanted to set the Content-Length programmatically...
[req setValue:[NSString stringWithFormat:@"%d", data.length]
    forHTTPHeaderField:@"Content-Length"];
```

# 22

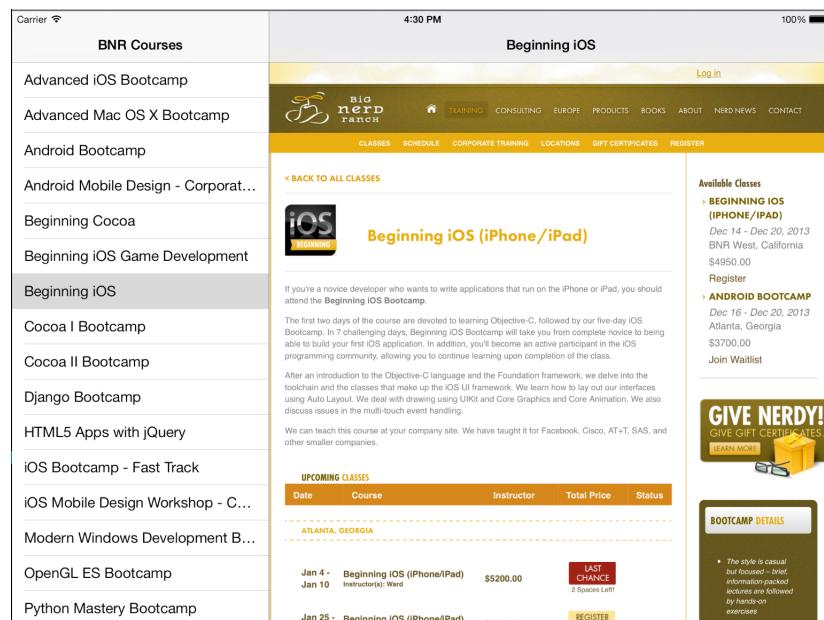
## UISplitViewController

O iPhone e o iPod touch têm uma quantidade limitada de espaço na tela. Considerando o pouco espaço na tela, quando apresentamos uma interface do tipo drill-down (detalhamento), uma **UINavigationController** é usada para alternar entre uma lista de itens e uma visão detalhada de um item.

O iPad, por outro lado, tem bastante espaço na tela para apresentar ambas as visões usando uma classe embutida chamada **UISplitViewController**. A **UISplitViewController** é uma classe apenas do iPad que apresenta dois controladores de visão em uma relação mestre-detalhes. O controlador de visão mestre ocupa uma pequena faixa do lado esquerdo da tela, e o controlador de visão de detalhes ocupa o restante da tela.

Neste capítulo, você fará com que o Nerdfeed apresente seus controladores de visão em um controlador de visão dividida quando estiver sendo executado em um iPad (Figure 22.1). Você também tornará o Nerdfeed um aplicativo universal, e fará com que ele continue usando uma **UINavigationController** quando estiver sendo executado em um iPhone.

Figure 22.1 Nerdfeed com **UISplitViewController**



### Divisão do Nerdfeed

A criação de uma **UISplitViewController** é simples agora que você já aprendeu sobre controladores de navegação e controladores de barra de guias. Quando inicializa um controlador de visão dividida, você passa para ele um array de controladores de visão, exatamente como se faz com um controlador de barra de guias. Porém, o array de um controlador de visão dividida limita-se a dois controladores de visão: um controlador de visão mestre e um controlador de visão de detalhes. A ordem dos controladores de visão no array determina suas funções na visão dividida; a primeira entrada é o controlador de visão mestre, e a segunda é o controlador de visão de detalhes.

Abra o `Nerdfeed.xcodeproj` no Xcode. Depois, abra o `BNRAppDelegate.m`.

No `application:didFinishLaunchingWithOptions:`, verifique se o dispositivo é um iPad antes de instanciar uma `UISplitViewController`. A classe `UISplitViewController` não existe no iPhone, e tentar criar uma instância de `UISplitViewController` causa uma exceção.

```
- (BOOL)application:(UIApplication *)application
    didFinishLaunchingWithOptions:(NSDictionary *)launchOptions
{
    self.window = [[UIWindow alloc] initWithFrame:[[UIScreen mainScreen] bounds]];

    BNRCoursesViewController *lvc =
        [[BNRCoursesViewController alloc] initWithStyle:UITableViewStylePlain];

    UINavigationController *masterNav =
        [[UINavigationController alloc] initWithRootViewController:lvc];

    BNRWebViewController *wvc = [[BNRWebViewController alloc] init];
    lvc.webViewController = wvc;

    self.window.rootViewController = masterNav;

    // Check to make sure we are running on the iPad
    if ([UIDevice currentDevice].userInterfaceIdiom == UIUserInterfaceIdiomPad) {

        // webViewController must be in navigation controller; you will see why later
        UINavigationController *detailNav =
            [[UINavigationController alloc] initWithRootViewController:wvc];

        UISplitViewController *svc = [[UISplitViewController alloc] init];

        // Set the delegate of the split view controller to the detail VC
        // You will need this later - ignore the warning for now
        svc.delegate = wvc;

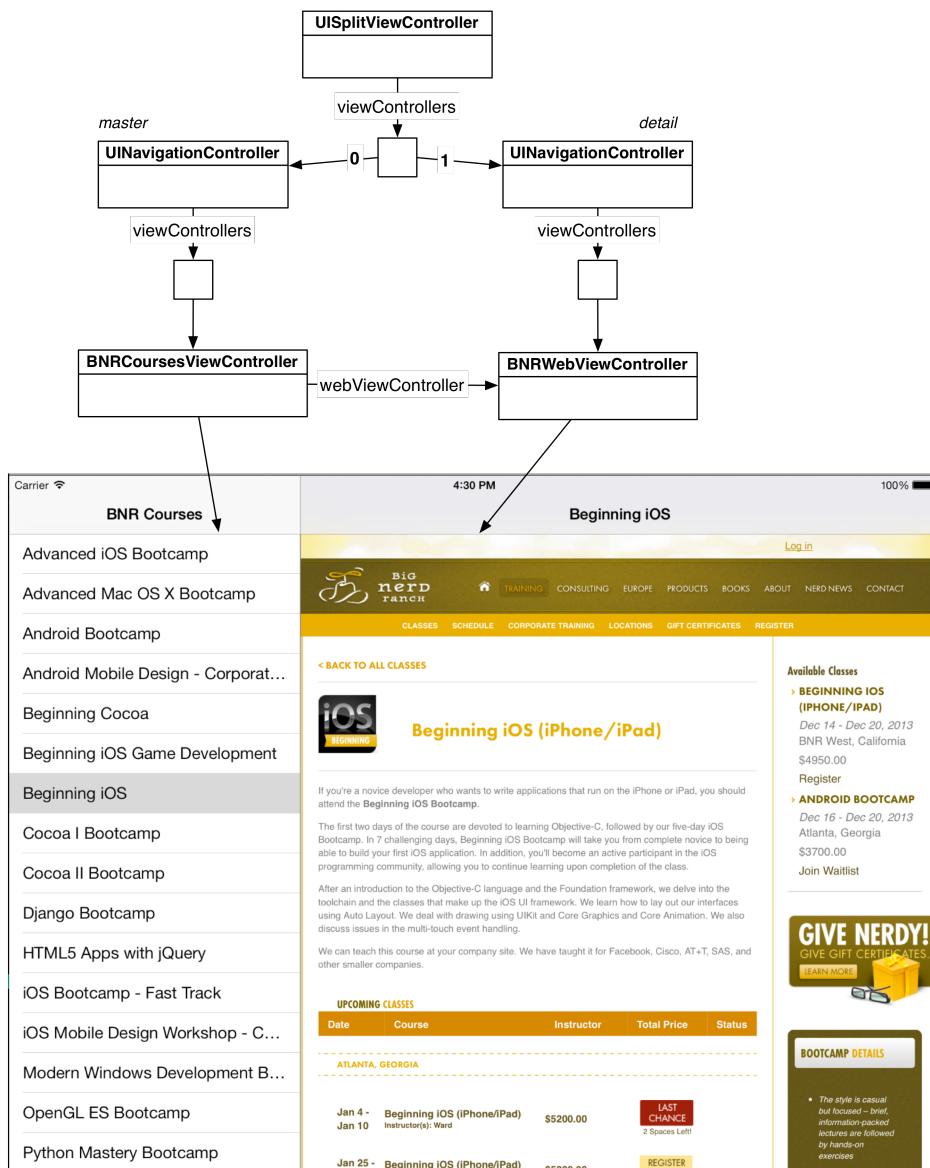
        svc.viewControllers = @[masterNav, detailNav];

        // Set the root view controller of the window to the split view controller
        self.window.rootViewController = svc;
    } else {
        // On non-iPad devices, just use the navigation controller
        self.window.rootViewController = masterNav;
    }

    self.window.backgroundColor = [UIColor whiteColor];
    [self.window makeKeyAndVisible];
    return YES;
}
```

Ao colocar o código `UISplitViewController` dentro de uma declaração `if` nesse método, estamos abrindo caminho para tornar o Nerdfeed um aplicativo universal. Além disso, agora você pode ver por que criou a instância de `BNRWebViewController` aqui em vez de seguir o caminho típico de criar um controlador de visão de detalhes dentro da implementação do controlador de visão raiz. Um controlador de visão dividida precisa ter tanto o controlador de visão mestre quanto o controlador de visão de detalhes quando é criado. O diagrama do controlador de visão dividida do Nerdfeed é mostrado na Figure 22.2.

Figure 22.2 Diagrama do controlador de visão dividida

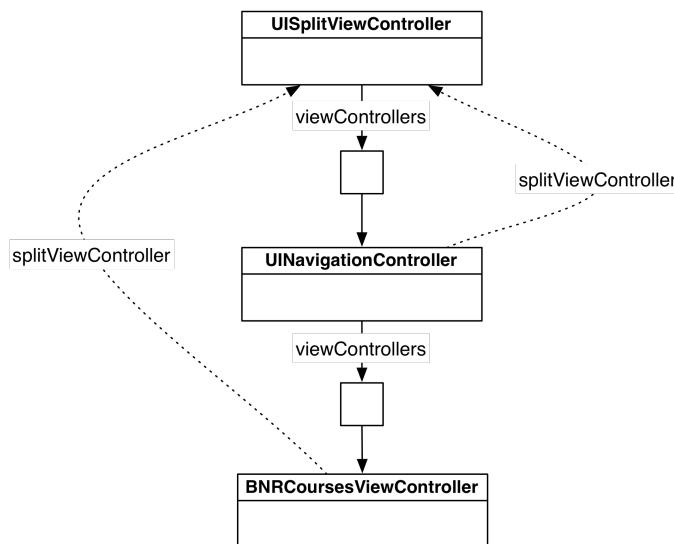


Compile e execute o aplicativo. Talvez você não veja nada ainda se estiver no modo retrato; no entanto, se você girar o dispositivo para paisagem, verá os dois controladores de visão na tela. É assim que o **UISplitViewController** funciona: no modo paisagem, o controlador de visão mestre é mostrado em uma pequena faixa do lado esquerdo da tela, e o controlador de visão de detalhes ocupa o restante da tela.

Mas você ainda não terminou. Se tocar em uma linha no controlador de visão de lista, o controlador de visão web não aparecerá no painel de detalhes como você queria. Em vez disso, ele é colocado no painel mestre e substitui o controlador de visão de lista. Para resolver esse problema, quando uma linha é tocada, você precisa verificar se a **BNRCoursesViewController** é um membro de um controlador de visão dividida e, se for, executar uma ação diferente.

Você pode enviar a mensagem **splitViewController** para qualquer **UIViewController**, e se esse controlador de visão fizer parte de um controlador de visão dividida, ele retornará um ponteiro para o controlador de visão dividida (Figure 22.3). Caso contrário, ele retorna `nil`. Os controladores de visão são inteligentes: um controlador de visão retorna esse ponteiro se ele for membro do array do controlador de visão dividida ou se pertencer a outro controlador que seja membro do array de um controlador de visão dividida (como é o caso tanto da **BNRCoursesViewController** quanto da **BNRWebViewController**).

Figure 22.3 propriedade splitViewController da UIViewController



No BNRCoursesViewController.m, localize o método `tableView:didSelectRowAtIndexPath:`. No topo desse método, verifique se existe um controlador de visão dividida antes de colocar a `BNRWebViewController` na pilha de navegação.

```

- (void)tableView:(UITableView *)tableView
    didSelectRowAtIndexPath:(NSIndexPath *)indexPath
{
    NSDictionary *course = self.courses[indexPath.row];
    NSURL *URL = [NSURL URLWithString:course[@"url"]];

    self.webViewController.title = course[@"title"];
    self.webViewController.URL = URL;
    [self.navigationController pushViewController:self.webViewController
                                         animated:YES];

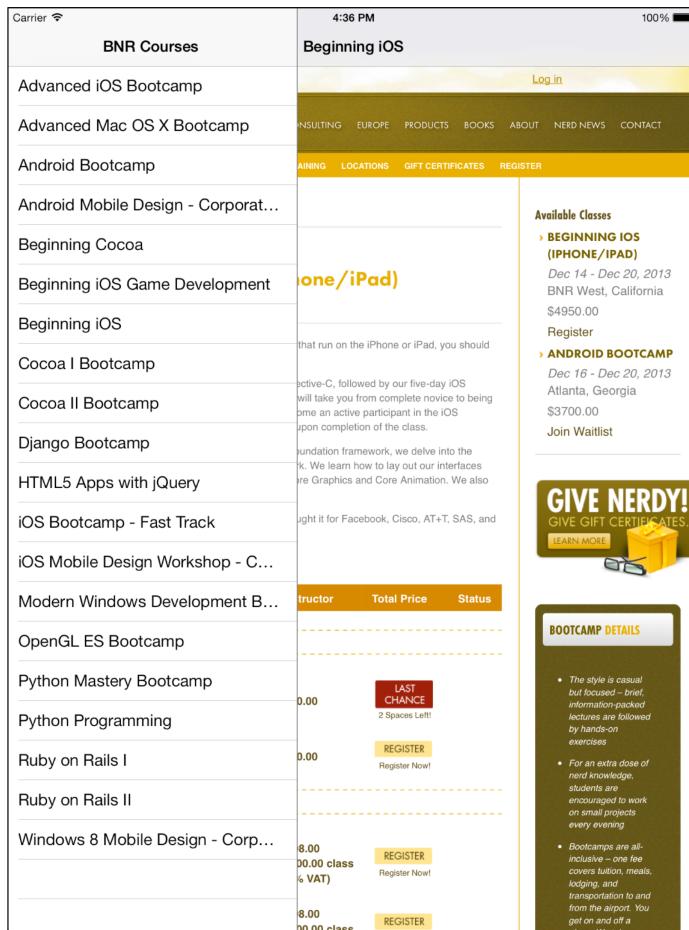
    if (!self.splitViewController) {
        [self.navigationController pushViewController:self.webViewController
                                         animated:YES];
    }
}
  
```

Agora, se a `BNRCoursesViewController` não estiver em um controlador de visão dividida, você pressupõe que o dispositivo não é um iPad e `BNRCoursesViewController` empurra `BNRWebViewController` na pilha do controlador de navegação. Se a `BNRCoursesViewController` estiver em um controlador de visão dividida, então o `UISplitViewController` é quem vai colocar `BNRWebViewController` na tela.

Compile e execute o aplicativo novamente. Rotacione para o modo paisagem e toque em uma das linhas. A página web da classe é agora carregada no painel de detalhes.

## Exibição do controlador de visão mestre no modo retrato

Quando no modo retrato, o controlador de visão mestre fica desaparecido. Seria bom se você conseguisse ver o controlador de visão mestre para selecionar uma nova postagem da lista sem ter que girar o dispositivo. A `UISplitViewController` deixa você fazer exatamente isso, oferecendo seu delegate com uma `UIBarButton Item`. Tocar nesse botão exibe o controlador de visão mestre em uma `UIPopoverController` (Figure 22.4) especializada.

Figure 22.4 Controlador de visão mestre em **UIPopoverController**

No seu código, sempre que um controlador de visão de detalhes foi dado ao controlador de visão dividida, aquele controlador de visão de detalhes foi configurado como delegate do controlador de visão dividida. Como delegate, o controlador de visão de detalhes obtém um ponteiro para **UIBarButtonItem** quando rotacionado para o modo retrato.

No **BNRWebViewController.h**, adicione essa declaração:

```
@interface BNRWebViewController : UIViewController <UISplitViewControllerDelegate>
```

Compile e execute o aplicativo. O comportamento será o mesmo, mas não haverá nenhum aviso.

No **BNRWebViewController.m**, implemente o método delegate a seguir para colocar o item de botão de barra no item de navegação de **BNRWebViewController**.

```
- (void)splitViewController:(UISplitViewController *)svc
    willHideViewController:(UIViewController *)aViewController
    withBarButtonItem:(UIBarButtonItem *)barButtonItem
    forPopoverController:(UIPopoverController *)pc
{
    // If this bar button item does not have a title, it will not appear at all
    barButtonItem.title = @"Courses";

    // Take this bar button item and put it on the left side of the nav item
    self.navigationItem.leftBarButtonItem = barButtonItem;
}
```

Observe que você definiu explicitamente o título do botão. Se o botão não tiver um título, ele simplesmente não aparecerá. (Se o **navigationItem** do controlador de visão mestre tiver um título, o botão será automaticamente definido com o mesmo título.)

Compile e execute o aplicativo. Gire para o modo retrato, e você verá o item de botão de barra aparecendo do lado esquerdo da barra de navegação. Toque nesse botão, e a visão do controlador de visão mestre aparecerá em uma **UIPopoverController**.

Esse item de botão de barra é o motivo pelo qual você sempre coloca o controlador de visão de detalhes dentro de um controlador de navegação. Você não precisa usar um controlador de navegação para colocar um controlador de visão em um controlador de visão dividida, mas isso facilita muito o uso do item de botão de barra. (Se você não usa um controlador de navegação, pode instanciar a sua própria **UINavigationBar** ou **UIToolbar** para armazenar o item de botão de barra e adicioná-lo como subvisão da visão da **BNRWebViewController**.)

Restam ainda dois probleminhas com seu botão Courses. Em primeiro lugar, quando o dispositivo é rotacionado de volta para o modo paisagem, o botão continua lá. Para removê-lo, o delegate precisa responder a outra mensagem de **UISplitViewController**. Implemente esse método delegate no **BNRWebViewController.m**.

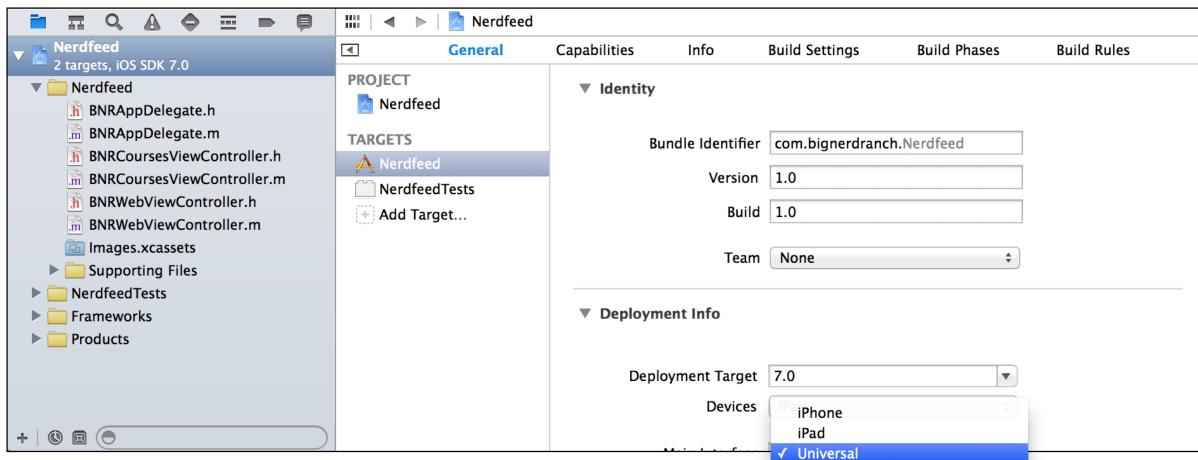
```
- (void)splitViewController:(UISplitViewController *)svc
    willShowViewController:(UIViewController *)aViewController
    invalidatingBarButtonItem:(UIBarButtonItem *)barButtonItem
{
    // Remove the bar button item from the navigation item
    // Double check that it is the correct button, even though we know it is
    if (barButtonItem == self.navigationItem.leftBarButtonItem) {
        self.navigationItem.leftBarButtonItem = nil;
    }
}
```

Compile e execute o aplicativo. O botão Courses agora irá aparecer e desaparecer conforme você alterna entre os modos retrato e paisagem.

## Universalização do Nerdfeed

Você criou o Nerdfeed como uma aplicação exclusiva de iPad; agora vai universalizá-la. Selecione o projeto Nerdfeed no navegador de projetos. Na área do editor, escolha o destino Nerdfeed e depois a guia General.

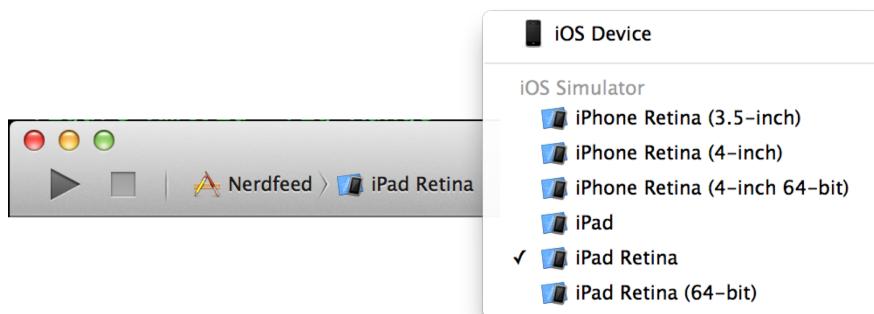
Figure 22.5 Universalização do Nerdfeed



No menu pop-up Devices, selecione Universal.

O aplicativo é agora universal. Você pode testá-lo compilando e executando novamente em um simulador e depois no outro.

Figure 22.6 Alteração de simuladores



Há dois motivos para o processo de universalização ter sido fácil para o Nerdfeed. Lembrar-se desses motivos ajudará quando você estiver escrevendo seus próprios aplicativos.

- Conforme você constrói o Nerdfeed, tenha cuidado com as diferenças entre dispositivos nas classes utilizadas. Por exemplo, sabendo que uma **UISplitViewController** não existe no iPhone ou iPod touch, você incluiu uma interface alternativa para esses dispositivos. No geral, quando se usa uma classe fornecida pela Apple, deve-se ler a discussão sobre essa classe na documentação. Isso lhe dará dicas sobre a disponibilidade da classe e de seus métodos nos diferentes dispositivos.
- O Nerdfeed é ainda um aplicativo relativamente simples. É sempre mais fácil universalizar um aplicativo no começo do desenvolvimento. Conforme ele cresce, os detalhes ficam perdidos em uma grande pilha de códigos. Encontrar e corrigir os problemas enquanto o código está sendo escrito é muito mais fácil do que retornar a eles depois. Os detalhes são difíceis de encontrar, e há o risco de estragar o que estava funcionando antes.



# 23

## Core Data

Ao decidir entre abordagens para salvar e carregar aplicativos iOS, a primeira pergunta geralmente é: “Local ou remotamente?” Se quiser salvar dados em um servidor remoto, isso é normalmente feito com um serviço web. Vamos pressupor que você queira armazenar dados localmente. A próxima pergunta geralmente é: “Arquivamento ou Core Data?”

No momento, o Homepwner usa arquivamento codificado para salvar dados de itens no sistema de arquivos. A maior desvantagem do arquivamento é a sua natureza de tudo ou nada: para acessar qualquer coisa no arquivamento, você precisa desarquivar todo o arquivo; para salvar qualquer alteração, você precisa regravar todo o arquivo. O Core Data, por outro lado, permite recuperar um pequeno subgrupo de objetos armazenados. E caso altere qualquer um dos objetos, você pode atualizar apenas aquela parte do arquivo. Essa obtenção, atualização, exclusão e inserção incremental pode melhorar radicalmente o desempenho de seu aplicativo quando você tem vários objetos de modelo sendo transferidos entre o sistema de arquivos e a RAM.

### Mapeamento objeto-relacional

Core Data é uma estrutura que oferece *mapeamento objeto-relacional*. Em outras palavras, o Core Data pode transformar objetos Objective-C em dados que são armazenados em um arquivo de banco de dados SQLite e vice-versa. O SQLite é um banco de dados relacional armazenado em um único arquivo (teoricamente, o SQLite é a biblioteca que gerencia o arquivo do banco de dados, mas utilizamos a palavra com o significado tanto de arquivo quanto de biblioteca). É importante notar que o SQLite não é um servidor de banco de dados relacional completo, como Oracle, MySQL ou SQLServer, que são seus próprios aplicativos aos quais os clientes podem se conectar através de uma rede.

O Core Data nos fornece a capacidade de obter e armazenar dados em um banco de dados relacional sem precisar conhecer o SQL. No entanto, é preciso de fato entender um pouco como os bancos de dados relacionais funcionam. Este capítulo lhe dará este entendimento conforme você substitui o arquivamento codificado pelo Core Data na Homepwner do `BNRItemStore`.

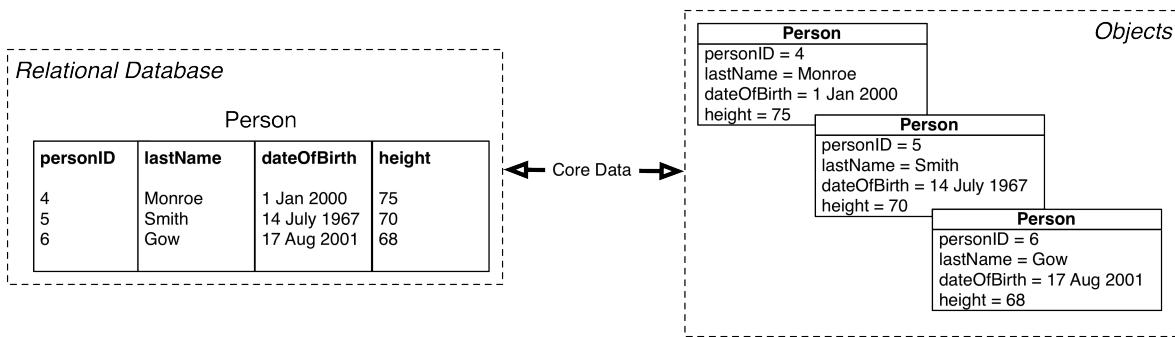
### Mudança do Homepwner para o Core Data

O seu aplicativo Homepwner atualmente utiliza o arquivamento para salvar e recarregar seus dados. Para um modelo de objetos de tamanho moderado (digamos, menos de 1000 objetos), tudo bem. À medida que o seu modelo de objetos aumenta, no entanto, você vai querer ser capaz de fazer buscas e atualizações incrementais e o Core Data pode fazer isso.

### O arquivo do modelo

Em um banco de dados relacional, temos algo chamado *tabela*. Uma tabela representa algum tipo; você pode ter uma tabela de pessoas, uma tabela de compras de cartão de crédito ou uma tabela de imóveis à venda. Toda tabela tem várias colunas para conter informações sobre esse tipo. A tabela que representa pessoas pode ter colunas para o sobrenome, a data de nascimento e a altura das pessoas. Cada linha da tabela representa uma única pessoa.

Figure 23.1 Papel do Core Data

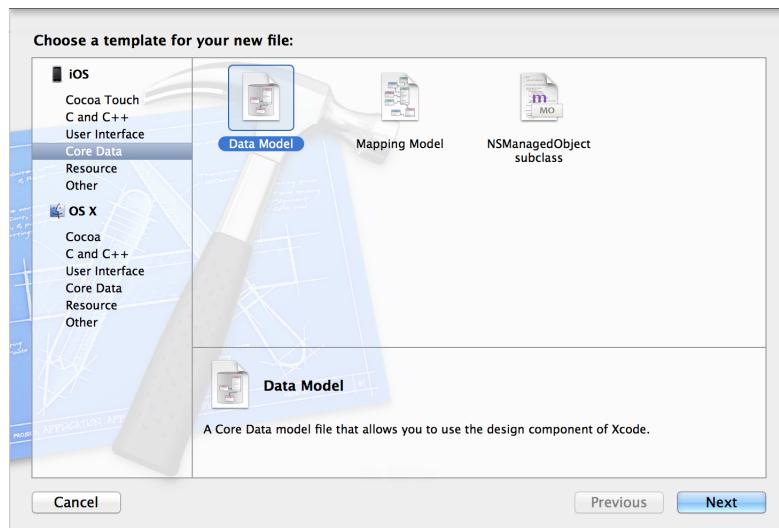


Essa organização se adequa bem ao Objective-C. Cada tabela é como uma classe do Objective-C. Cada coluna é uma das propriedades da classe. Cada linha é uma instância dessa classe. Portanto, o trabalho do Core Data é mover os dados de e para essas duas representações (Figure 23.1).

O Core Data usa terminologia diferente para descrever essas ideias: uma tabela/classe é chamada de *entidade* e as colunas/propriedades são chamadas de *atributos*. Um arquivo do modelo do Core Data é a descrição de cada entidade juntamente com os seus atributos no seu aplicativo. No Homeowner, você vai descrever uma entidade **BNRItem** em um arquivo do modelo e conferir atributos a ela, como `itemName`, `serialNumber` e `valueInDollars`.

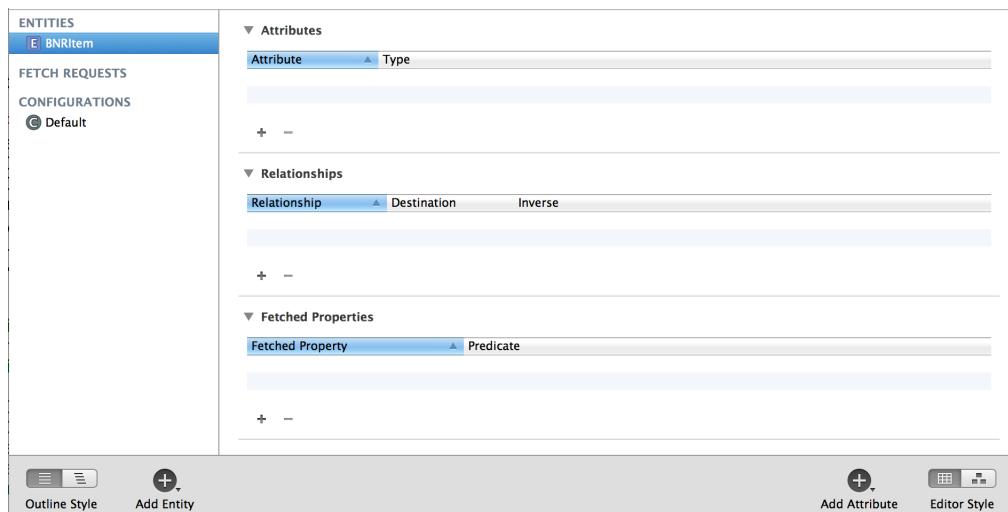
Abra o `Homeowner.xcodeproj`. No menu `File`, crie um novo arquivo. Selecione Core Data na seção iOS e crie um novo Data Model. Nomeie-o como `Homeowner`.

Figure 23.2 Criação de arquivo do modelo



Isso vai criar um arquivo `Homeowner.xcdatamodeld` e adicioná-lo ao seu projeto. Selecione esse arquivo no navegador de projetos e a área do editor vai revelar a interface do usuário para manipular o arquivo do modelo do Core Data.

Localize o botão **Add Entity** na parte inferior esquerda da janela e clique nele. Uma nova Entity aparecerá na lista de entidades, no lado esquerdo da tabela. Clique duas vezes nessa entidade e altere seu nome para **BNRItem** (Figure 23.3).

Figure 23.3 Criação da entidade **BNRItem**

Agora a sua entidade **BNRItem** precisa de atributos. Lembre-se de que estes serão as propriedades da classe **BNRItem**. Os atributos necessários estão listados abaixo. Para cada atributo, clique no botão + na seção Attributes e edite os valores de Attribute e Type:

- `itemName` é uma String
- `serialNumber` é uma String
- `valueInDollars` é um Integer 32
- `dateCreated` é uma Date
- `itemKey` é uma String
- `thumbnail` é um Transformable (é uma **UIImage**, mas esta não é uma das possibilidades. Conversaremos sobre Transformable em breve).

É necessário adicionar mais um atributo. No Homeowner, os usuários podem ordenar os itens, mudando suas posições na visão da tabela. Arquivar itens em um array respeita naturalmente essa ordenação. No entanto, tabelas relacionais não ordenam suas linhas. Em vez disso, ao obter um conjunto de linhas, você especifica a sua ordenação usando um dos atributos (“Obtenha para mim todos os objetos **BNREmployee**, ordenados por `lastName`.”).

Para manter a ordem dos itens, é preciso criar um atributo para registrar a posição de cada item na visão da tabela. Assim, quando obtiver itens, você pode solicitar que sejam ordenados por este atributo (você também precisará atualizar esse atributo quando os itens forem reordenados). Crie este último atributo: chame-o de `orderingValue` e torne-o um Double.

O Core Data é capaz de manter apenas certos tipos de dados em seu armazenamento, e **UIImage** não é um desses tipos. Em vez disso, você declarou a `thumbnail` como *transformável*. Com um atributo transformável, o Core Data converterá o objeto em **NSData** ao salvá-lo, e converterá a **NSData** de volta para o objeto original ao carregá-la do sistema de arquivos. Para que o Core Data faça isso, você terá de fornecer a ele uma subclasse **NSValueTransformer** que trata dessas conversões.

Crie uma nova classe denominada **BNRImageTransformer**, que é uma subclasse de **NSValueTransformer**. Abra o `BNRImageTransformer.m` e sobrescreva os métodos necessários para transformar a **UIImage** de e para **NSData**.

```

@implementation BNRImageTransformer

+ (Class)transformedValueClass
{
    return [NSData class];
}

- (id)transformedValue:(id)value
{
    if (!value) {
        return nil;
    }

    if ([value isKindOfClass:[NSData class]]) {
        return value;
    }

    return UIImagePNGRepresentation(value);
}

- (id)reverseTransformedValue:(id)value
{
    return [UIImage imageWithData:value];
}

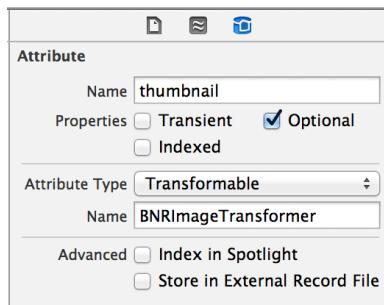
@end

```

A implementação de **BNRImageTransformer** é bastante objetiva. O método de classe **transformedValueClass** diz ao transformador que tipo de objeto receberá do método **transformedValue:**. O método **transformedValue:** será chamado quando sua variável transformável estiver prestes a ser salva no sistema de arquivos e for esperado um objeto que possa ser salvo no Core Data. Neste exemplo, o argumento para o método será uma **UIImage** e retornará uma instância de **NSData**. Finalmente, o método **reverseTransformedValue:** é chamado quando os dados das miniaturas são carregados pelo sistema de arquivos; sua implementação criará a **UIImage** com base na **NSData** que foi armazenada. Quando esse arquivo for criado, o Core Data deverá saber usar essa classe ao trabalhar com a thumbnail.

Abra o `Homepwner.xcdatamodeld` e selecione a entidade **BNRItem**. Selecione thumbnail na lista **Attributes**, clique na guia no seletor de inspetores para exibir o *inspetor de modelo de dados*. Substitua o texto do placeholder **Value Transformer Name** no segundo campo **Name** por **BNRImageTransformer** (Figure 23.4).

Figure 23.4 Atualização do nome do transformador de valor para o atributo das miniaturas



A essa altura, o seu arquivo do modelo é suficiente para salvar e carregar itens. No entanto, uma das vantagens de se usar o Core Data é que as entidades podem ser relacionadas entre si, portanto, você adicionará uma nova entidade chamada **BNRAssetType** que descreve uma categoria de itens. Por exemplo, um quadro pode ser do tipo de ativo Arte. **BNRAssetType** será uma entidade no arquivo do modelo e cada linha dessa tabela será mapeada para um objeto do Objective-C em tempo de execução.

No `Homepwner.xcdatamodeld`, adicione outra entidade chamada **BNRAssetType**. Dê a ela um atributo chamado **label** do tipo String. Esse será o nome da categoria que **BNRAssetType** representa.

Figure 23.5 Criação da entidade **BNRAssetType**

The screenshot shows the Xcode Model Editor interface. On the left, under 'ENTITIES', there are two entities: 'BNRAssetType' (selected) and 'BNRItem'. On the right, under 'Attributes', there is one attribute named 'label' with a type of 'String'. Below the attributes is a table for 'Relationships'.

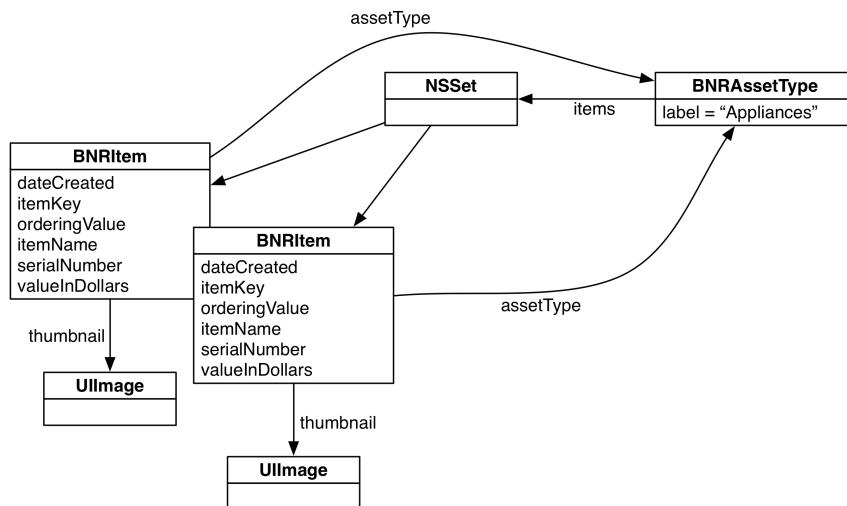
Agora, você precisa estabelecer os relacionamentos entre **BNRAssetType** e **BNRItem**. Os relacionamentos entre entidades são representados por ponteiros entre objetos. Há dois tipos de relacionamentos: *um para um* e *um para vários*.

Quando uma entidade tem um relacionamento *um para um*, cada instância dessa entidade terá um ponteiro para uma instância da entidade com a qual tem um relacionamento. A entidade **BNRItem** terá um relacionamento *um para um* com a entidade **BNRAssetType**. Assim, uma instância de **BNRItem** terá um ponteiro para uma instância de **BNRAssetType**.

Quando uma entidade tem um relacionamento *um para vários*, cada instância desta entidade tem um ponteiro para um **NSSet**. Este conjunto contém as instâncias da entidade com a qual ela tem uma relação. A entidade **BNRAssetType** terá um relacionamento *um para vários* com **BNRItem**, pois muitas instâncias de **BNRItem** podem ter a mesma **BNRAssetType**. Assim, um objeto **BNRAssetType** terá um ponteiro para um conjunto com todos os objetos **BNRItem** que têm o seu tipo de ativo.

Com esses relacionamentos configurados, você pode pedir a um objeto **BNRAssetType** o conjunto de objetos **BNRItem** que se encaixa em sua categoria e perguntar a uma **BNRItem** em que **BNRAssetType** ela se encaixa. A Figure 23.6 apresenta um diagrama dos relacionamentos entre **BNRAssetType** e **BNRItem**.

Figure 23.6 Entidades no Homepwner



Vamos adicionar esses relacionamentos ao arquivo do modelo. Selecione a entidade **BNRAssetType** e, em seguida, clique no botão **+** na seção Relationships. Dê o nome de **items** a esse relacionamento na coluna Relationship. Em seguida, selecione **BNRItem** na coluna Destination. No inspetor de modelo de dados, mude o menu suspenso Type de To One para To Many (Figure 23.7).

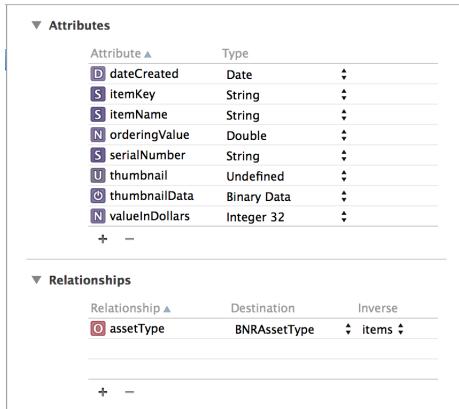
Figure 23.7 Criação do relacionamento de **items**

The screenshot shows the Xcode Model Editor interface. On the left, under 'Relationships', there is a new entry 'items' with 'BNRItem' as the 'Destination' and 'assetType' as the 'Inverse'. The 'Relationship' inspector on the right shows the following settings:

- Name: items
- Type: To Many
- Destination: BNRItem
- Inverse: assetType
- Delete Rule: Nullify
- Count: Unbounded
- Advanced: Index in Spotlight, Store in External Record File

Agora, retorne para a entidade **BNRItem**. Adicione um relacionamento chamado **assetType** e escolha **BNRAssetType** como seu destino. Na coluna **Inverse**, selecione **items** (Figure 23.8).

Figure 23.8 Criação do relacionamento **assetType**



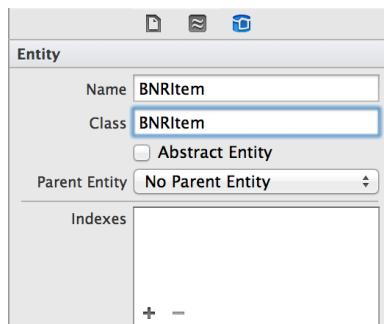
## NSManagedObject e subclasses

Quando um objeto é obtido com o Core Data, sua classe, por padrão, é **NSManagedObject**. **NSManagedObject** é uma subclasse de **NSObject** que sabe como cooperar com o resto do Core Data. A **NSManagedObject** funciona um pouco como um dicionário: contém um par chave-valor para cada propriedade (atributo ou relacionamento) na entidade.

A **NSManagedObject** é um pouco mais do que um recipiente de dados. Se você precisa que seus objetos de modelo *fazam* algo além de conter dados, é preciso criar subclasses de **NSManagedObject**. Assim, em seu arquivo do modelo, você especifica que essa entidade é representada por instâncias de sua subclasse, não a **NSManagedObject** padrão.

Selecione a entidade **BNRItem**. Exiba o inspetor de modelo de dados e altere o campo **Class** para **BNRItem**, como mostrado na Figure 23.9. Agora, quando uma entidade **BNRItem** é obtida com o Core Data, o tipo desse objeto será **BNRItem**. (Instâncias de **BNRAssetType** ainda serão do tipo **NSManagedObject**.)

Figure 23.9 Alteração da classe de uma entidade



Há um problema: a classe **BNRItem** já existe e ela não herda de **NSManagedObject**. Alterar a superclasse da **BNRItem** existente para **NSManagedObject** exigirá modificações significativas. Desse modo, a solução mais fácil é remover os arquivos da classe **BNRItem** atual, fazer com que o Core Data gere uma nova classe **BNRItem** e, em seguida, adicionar seus métodos de comportamento de volta aos arquivos da nova classe.

No Finder, arraste tanto o **BNRItem.h** quanto o **BNRItem.m** para sua área de trabalho por questões de segurança. Em seguida, no Xcode, exclua esses dois arquivos do navegador de projetos. (Eles estarão em vermelho, depois de movidos.)

Agora, abra o Homepwner.xcdatamodeld novamente e selecione a entidade **BNRItem**. Em seguida, selecione New File... no menu New.

Na seção iOS, selecione Core Data, escolha a opção NSManagedObject subclass e clique em Next. A caixa de seleção dos dados de Homepwner já deve estar marcada. Caso não esteja, marque a caixa e clique em Next. Na próxima tela, certifique-se de que a **BNRItem** esteja marcada e clique em Next uma última vez. Finalmente, clique em Create para gerar arquivos da subclasse **NSManagedObject**.

Xcode gerará novos arquivos BNRItem.h e BNRItem.m. Abra o BNRItem.h e veja o que o Core Data gerou. Altere o tipo da propriedade da thumbnail para **UIImage** e adicione a declaração de método de **BNRItem** anterior. Por padrão, o Xcode gera as propriedades como objetos, portanto seus ints são agora instâncias de **NSNumber**. Altere orderingValue para double e valueInDollars para int.

```
#import <Foundation/Foundation.h>
#import CoreData;

@interface BNRItem : NSManagedObject

@property (nonatomic, strong) NSDate * dateCreated;
@property (nonatomic, strong) NSString * itemKey;
@property (nonatomic, strong) NSString * itemName;
@property (nonatomic, strong) NSNumber * orderingValue;
@property (nonatomic) double orderingValue;
@property (nonatomic, strong) NSString * serialNumber;
@property (nonatomic, strong) UNKNOWN_TYPE thumbnail;
@property (nonatomic, strong) UIImage *thumbnail;
@property (nonatomic, strong) NSData * thumbnailData;
@property (nonatomic, strong) NSNumber * valueInDollars;
@property (nonatomic) int valueInDollars;
@property (nonatomic, strong) NSManagedObject *assetType;

- (void)setThumbnailFromImage:(UIImage *)image;

@end
```

(O Xcode pode ter criado as propriedades **strong** como retain. Elas representam as mesmas coisas; antes da ARC, as propriedades **strong** eram chamadas de **retain** e nem todas as partes das ferramentas foram atualizadas com a nova terminologia.)

Copie o método **setThumbnailFromImage:** de seu antigo BNRItem.m para o novo:

```
- (void)setThumbnailFromImage:(UIImage *)image
{
    CGSize origImageSize = image.size;
    CGRect newRect = CGRectMake(0, 0, 40, 40);

    float ratio = MAX(newRect.size.width / origImageSize.width,
                      newRect.size.height / origImageSize.height);

    UIGraphicsBeginImageContextWithOptions(newRect.size, NO, 0.0);
    UIBezierPath *path = [UIBezierPath bezierPathWithRoundedRect:newRect
                                                          cornerRadius:5.0];
    [path addClip];

    CGRect projectRect;
    projectRect.size.width = ratio * origImageSize.width;
    projectRect.size.height = ratio * origImageSize.height;
    projectRect.origin.x = (newRect.size.width - projectRect.size.width) / 2.0;
    projectRect.origin.y = (newRect.size.height - projectRect.size.height) / 2.0;

    [image drawInRect:projectRect];

    UIImage *smallImage = UIGraphicsGetImageFromCurrentImageContext();
    self.thumbnail = smallImage;

    UIGraphicsEndImageContext();
}
```

Claro, quando você executar um aplicativo pela primeira vez, não haverá itens salvos ou tipos de ativos. Quando o usuário cria uma nova instância de **BNRItem**, ela é adicionada ao banco de dados. Quando objetos são adicionados ao banco de dados, lhes é enviada a mensagem **awakeFromInsert**. É aqui que você definirá as propriedades `dateCreated` de `itemKey` de uma **BNRItem**. Implemente **awakeFromInsert** no **BNRItem.m**.

```
- (void)awakeFromInsert
{
    [super awakeFromInsert];

    self.dateCreated = [NSDate date];

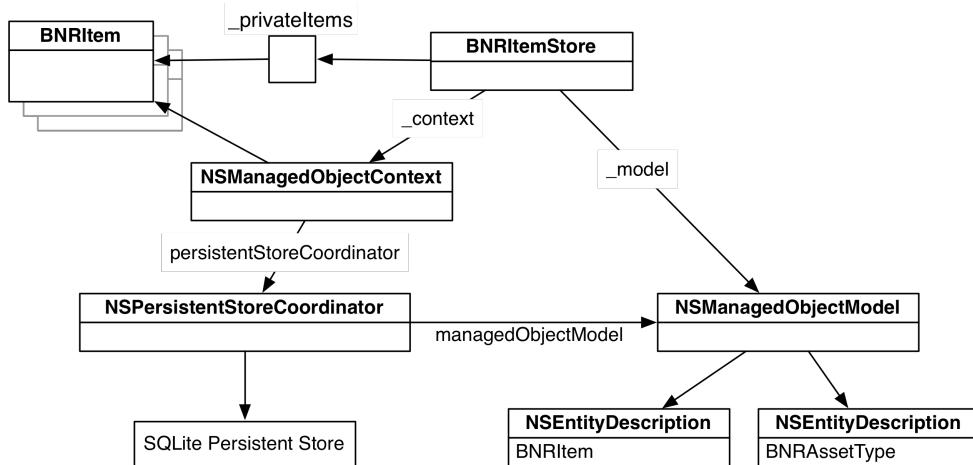
    // Create an NSUUID object - and get its string representation
    NSUUID *uuid = [[NSUUID alloc] init];
    NSString *key = [uuid UUIDString];
    self.itemKey = key;
}
```

Isso adiciona o comportamento extra do antigo inicializador designado de **BNRItem**. Compile o aplicativo para verificar se há erros de sintaxe, mas não o execute.

## Atualização de **BNRItemStore**

O portal por meio do qual você se comunica com o banco de dados é a **NSManagedObjectContext**. A **NSManagedObjectContext** usa uma **NSPersistentStoreCoordinator**. Você pede ao coordenador de armazenamento persistente que abra um banco de dados SQLite em um determinado nome de arquivo. O coordenador de armazenamento persistente usa o arquivo do modelo na forma de uma instância de **NSManagedObjectModel**. No Homepwner, esses objetos funcionarão com a **BNRItemStore**. Esses relacionamentos são exibidos na Figure 23.10.

Figure 23.10 **BNRItemStore** e **NSManagedObjectContext**



No **BNRItemStore.m**, importe o Core Data e adicione três propriedades à extensão da classe.

```
@import CoreData;

@interface BNRItemStore : NSObject
```

~~```
@property (nonatomic) NSMutableArray *privateItems;
@property (nonatomic, strong) NSMutableArray *allAssetTypes;
@property (nonatomic, strong) NSManagedObjectContext *context;
@property (nonatomic, strong) NSManagedObjectModel *model;
```~~

Depois, altere a implementação do `itemArchivePath` para retornar um caminho diferente que o Core Data usará para salvar dados.

```

- (NSString *)itemArchivePath
{
    NSArray *documentDirectories =
        NSSearchPathForDirectoriesInDomains(NSDocumentDirectory,
                                            NSUserDomainMask,
                                            YES);

    // Get one and only document directory from that list
    NSString *documentDirectory = [documentDirectories firstObject];

    return [documentDirectory stringByAppendingPathComponent:@"items.archive"];
    return [documentDirectory stringByAppendingPathComponent:@"store.data"];
}

```

Quando a **BNRItemStore** é inicializada, ela precisa definir a **NSManagedObjectContext** e uma **NSPersistentStoreCoordinator**. O coordenador de armazenamento persistente precisa saber duas coisas: “Quais são todas as minhas entidades, seus atributos e relacionamentos?” e “De onde estou carregando e onde estou salvando dados?” Para responder a essas perguntas, você precisa criar uma instância de **NSManagedObjectModel** para conter as informações de Homepwner.xcdatamodeld e inicializar o coordenador de armazenamento persistente com esse objeto. Em seguida, você criará a instância de **NSManagedObjectContext** e especificará que ela use esse coordenador de armazenamento persistente para salvar e carregar objetos.

No BNRItemStore.m, atualize o **initPrivate**.

```

- (instancetype)initPrivate
{
    self = [super init];
    if (self) {
        NSString *path = self.itemArchivePath;
        _privateItems = [NSKeyedUnarchiver unarchiveObjectWithFile:path];

        if (!_privateItems) {
            _privateItems = [[NSMutableArray alloc] init];
        }

        // Read in Homepwner.xcdatamodeld
        _model = [NSManagedObjectModel mergedModelFromBundles:nil];

        NSPersistentStoreCoordinator *psc =
            [[NSPersistentStoreCoordinator alloc] initWithManagedObjectModel:_model];

        // Where does the SQLite file go?
        NSString *path = self.itemArchivePath;
        NSURL *storeURL = [NSURL fileURLWithPath:path];

        NSError *error = nil;

        if (![psc addPersistentStoreWithType:NSSQLiteStoreType
                                      configuration:nil
                                         URL:storeURL
                                         options:nil
                                         error:&error]) {
            @throw [NSEException exceptionWithName:@"OpenFailure"
                                         reason:[error localizedDescription]
                                         userInfo:nil];
        }

        // Create the managed object context
        _context = [[NSManagedObjectContext alloc] init];
        _context.persistentStoreCoordinator = psc;
    }
    return self;
}

```

Antes, **BNRItemStore** gravaria toda a **NSMutableArray** de objetos **BNRItem** quando você solicitou que o salvamento fosse realizado usando arquivamento codificado. Agora, você fará com que ele envie a mensagem **save:** para a **NSManagedObjectContext**. O contexto atualizará todos os registros em **store.data** com todas as alterações desde a última vez em que foi salvo. No BNRItemStore.m, altere o **saveChanges**.

```
- (BOOL)saveChanges
{
    NSString *path = [self itemArchivePath];
    return [NSKeyedArchiver archiveRootObject:allItems
                                         toFile:[self itemArchivePath]];

    NSError *error;
    BOOL successful = [self.context save:&error];
    if (!successful) {
        NSLog(@"Error saving: %@", [error localizedDescription]);
    }
    return successful;
}
```

Lembre-se de que esse método já é chamado quando o aplicativo é movido para o segundo plano.

## NSFetchRequest e NSPredicate

Nesse aplicativo, você vai obter todos os itens no `store.data` quando precisar deles pela primeira vez. Para obter objetos de volta da `NSManagedObjectContext`, você precisa preparar e executar uma `NSFetchRequest`. Depois que uma solicitação de busca é executada, você terá um array de todos os objetos que correspondem aos parâmetros dessa solicitação.

Uma solicitação de busca precisa de uma descrição de entidade que defina de qual entidade você deseja obter objetos. Para obter instâncias de `BNRItem`, você especifica a entidade `BNRItem`. Você também pode definir os *descritores de classificação* para especificar a ordem dos objetos no array. Um descritor de classificação tem uma chave que mapeia para um atributo da entidade e um `BOOL` que indica se a ordem deve ser crescente ou decrescente. Você quer ordenar as instâncias de `BNRItem` retornadas por `orderingValue` em ordem crescente.

No `BNRItemStore.m`, defina um novo método, `loadAllItems`, para preparar e executar a solicitação de busca e salvar os resultados no array `allItems`.

```
- (void)loadAllItems
{
    if (!self.privateItems) {
        NSFetchRequest *request = [[NSFetchRequest alloc] init];

        NSEntityDescription *e = [NSEntityDescription entityForName:@"BNRItem"
                                         inManagedObjectContext:self.context];
        request.entity = e;

        NSSortDescriptor *sd = [NSSortDescriptor
                               sortDescriptorWithKey:@"orderingValue"
                               ascending:YES];
        request.sortDescriptors = @[sd];

        NSError *error;
        NSArray *result = [self.context executeFetchRequest:request error:&error];
        if (!result) {
            [NSError raise:@"Fetch failed"
                      format:@"Reason: %@", [error localizedDescription]];
        }

        self.privateItems = [[NSMutableArray alloc] initWithArray:result];
    }
}
```

Além disso, no `BNRItemStore.m`, envie esta mensagem para a `BNRItemStore` ao final de `initPrivate`.

```
_context.persistentStoreCoordinator = psc;
[self loadAllItems];
return self;
}
```

Você pode fazer a compilação para verificar se há erros de sintaxe.

Nesse aplicativo, você buscou imediatamente todas as instâncias da entidade **BNRItem**. Essa é uma simples solicitação. Em um aplicativo com um conjunto de dados muito maior, você obteria cuidadosamente apenas as instâncias que precisasse. Para obter instâncias seletivamente, você adiciona um *predicado* (uma **NSPredicate**) à sua solicitação de busca e apenas os objetos que satisfazem a esse predicado são retornados.

Um predicado contém uma condição que pode ser verdadeira ou falsa. Por exemplo, se você quisesse apenas itens que valem mais que \$50, criaria um predicado e o adicionaria à solicitação de busca desta forma:

```
NSPredicate *p = [NSPredicate predicateWithFormat:@"valueInDollars > 50"];
[request setPredicate:p];
```

A string de formatação para um predicado pode ser muito longa e complexa. O *Predicate Programming Guide* (Guia de programação de predicados) da Apple é uma discussão completa do que é possível.

Predicados também podem ser usados para filtrar o conteúdo de um array. Portanto, mesmo que já tivesse obtido o array **allItems**, você ainda poderia usar um predicado:

```
NSArray *expensiveStuff = [allItems filteredArrayUsingPredicate:p];
```

## Adicionar e excluir itens

Até agora, você conseguiu salvar e carregar, mas e quanto a adicionar e excluir? Quando o usuário desejar criar uma nova **BNRItem**, você não vai alocar e inicializar essa nova **BNRItem**. Em vez disso, você pedirá à **NSManagedObjectContext** que insira um novo objeto da entidade **BNRItem**. Ela retornará uma instância de **BNRItem**.

No **BNRItemStore.m**, edite o método **createItem**.

```
- (BNRItem *)createItem
{
    BNRItem *item = [[BNRItem alloc] init];
    double order;
    if ([self.allItems count] == 0) {
        order = 1.0;
    } else {
        order = [[self.privateItems lastObject] orderingValue] + 1.0;
    }
    NSLog(@"Adding after %d items, order = %.2f", [self.privateItems count], order);

    item.orderingValue = order;
    [self.privateItems addObject:item];
    return item;
}
```

Quando um usuário excluir uma **BNRItem**, você deve informar o contexto para que seja eliminado do banco de dados. No **BNRItemStore.m**, adicione o código a seguir ao **removeItem**:

```
- (void)removeItem:(BNRItem *)item
{
    NSString *key = item.itemKey;
    if (key) {
        [[BNRImageStore sharedStore] deleteImageForKey:key];
    }
    [self.context deleteObject:item];
    [self.privateItems removeObjectIdenticalTo:item];
}
```

## Reordenação de itens

A última funcionalidade que você precisa substituir por **BNRItem** é a capacidade de reordenar itens na **BNRItemStore**. Como o Core Data não lida com ordenação automaticamente, você precisa atualizar a `orderingValue` de **BNRItem** sempre que ela for movida na visão de tabela.

Isso se tornaria bastante complicado se a `orderingValue` fosse um inteiro: sempre que uma **BNRItem** fosse colocada em um novo índice, você teria de mudar a `orderingValue` de outros itens. É por isso que você criou `orderingValue` como double. Você pode pegar as `orderingValues` dos itens que virão antes e depois do item sendo movido, somá-los e dividir por dois. A nova `orderingValue` ficará diretamente entre os valores dos itens que o cercam.

No `BNRItemStore.m`, modifique `moveItemAtIndex:toIndex:` de forma que ele trate a reordenação de itens.

```
- (void)moveItemAtIndex:(NSInteger)fromIndex
                  toIndex:(NSInteger)toIndex
{
    if (fromIndex == toIndex) {
        return;
    }
    BNRItem *item = self.privateItems[fromIndex];

    [self.privateItems removeObjectAtIndex:fromIndex];
    [self.privateItems insertObject:item atIndex:toIndex];

    // Computing a new orderValue for the object that was moved
    double lowerBound = 0.0;

    // Is there an object before it in the array?
    if (toIndex > 0) {
        lowerBound = [self.privateItems[(toIndex - 1)] orderingValue];
    } else {
        lowerBound = [self.privateItems[1] orderingValue] - 2.0;
    }

    double upperBound = 0.0;

    // Is there an object after it in the array?
    if (toIndex < [self.privateItems count] - 1) {
        upperBound = [self.privateItems[(toIndex + 1)] orderingValue];
    } else {
        upperBound = [self.privateItems[(toIndex - 1)] orderingValue] + 2.0;
    }

    double newOrderValue = (lowerBound + upperBound) / 2.0;

    NSLog(@"%@", moving to order %f", newOrderValue);
    item.orderingValue = newOrderValue;
}
```

Finalmente, você pode compilar e executar seu aplicativo. É claro que o comportamento é o mesmo de sempre, mas agora está usando o Core Data.

## Adição de **BNRAssetTypes** ao Homepwner

No arquivo do modelo, você descreveu uma nova entidade, **BNRAssetType**, com a qual todos os itens terão uma relação um para um. Você precisa disponibilizar para o usuário uma maneira de definir a **BNRAssetType** de uma **BNRItem**. Além disso, a **BNRItemStore** precisará de uma maneira para buscar os tipos de ativos. (A criação de novas instâncias de **BNRAssetType** é apresentada como desafio no final deste capítulo.)

No `BNRItemStore.h`, declare um novo método.

```
- (NSArray *)allAssetTypes;
```

No `BNRItemStore.m`, defina esse método. Se esta é a primeira vez que o aplicativo é executado – e, portanto, não há objetos **BNRAssetType** no armazenamento – crie três tipos-padrão.

```

- (NSArray *)allAssetTypes
{
    if (!_allAssetTypes) {
        NSFetchedResultsController *request = [[NSFetchedResultsController alloc] init];
        NSEntityDescription *e = [NSEntityDescription entityForName:@"BNRAssetType"
                                inManagedObjectContext:self.context];
        request.entity = e;

        NSError *error = nil;
        NSArray *result = [self.context executeFetchRequest:request error:&error];
        if (!result) {
            [NSErrorException raise:@"Fetch failed"
                format:@"Reason: %@", [error localizedDescription]];
        }
        _allAssetTypes = [result mutableCopy];
    }

    // Is this the first time the program is being run?
    if ([_allAssetTypes count] == 0) {
        NSManagedObject *type;

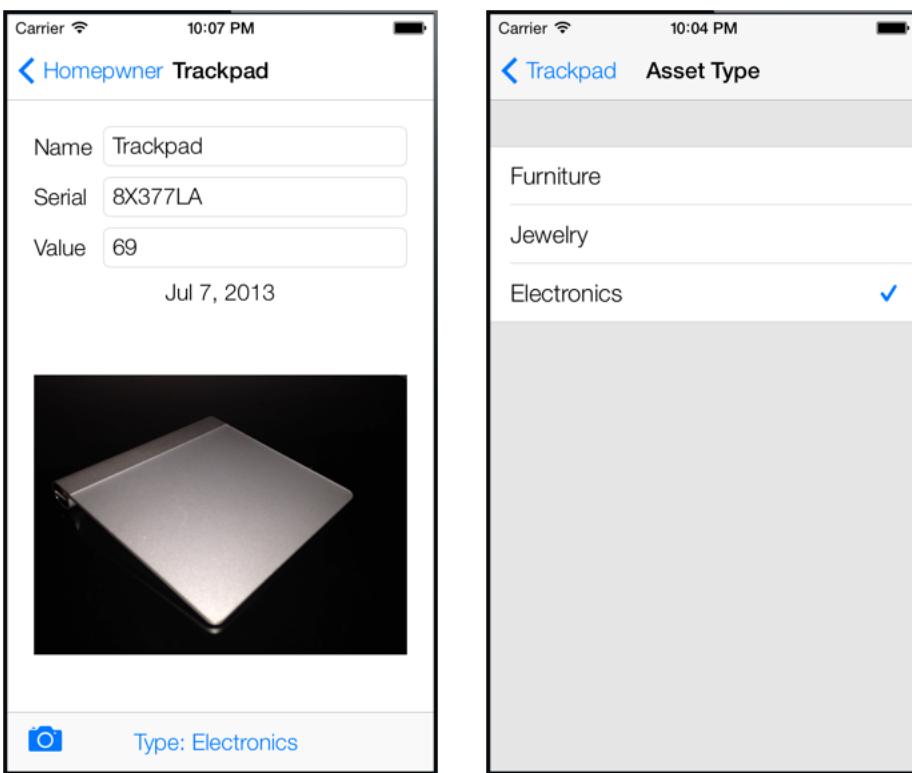
        type = [NSEntityDescription insertNewObjectForEntityForName:@"BNRAssetType"
                           inManagedObjectContext:self.context];
        [type setValue:@"Furniture" forKey:@"label"];
        [_allAssetTypes addObject:type];

        type = [NSEntityDescription insertNewObjectForEntityForName:@"BNRAssetType"
                           inManagedObjectContext:self.context];
        [type setValue:@"Jewelry" forKey:@"label"];
        [_allAssetTypes addObject:type];

        type = [NSEntityDescription insertNewObjectForEntityForName:@"BNRAssetType"
                           inManagedObjectContext:self.context];
        [type setValue:@"Electronics" forKey:@"label"];
        [_allAssetTypes addObject:type];
    }
    return _allAssetTypes;
}

```

Agora, você precisa mudar a interface do usuário para que ele possa ver e alterar a **BNRAssetType** da **BNRItem** na **BNRDetailViewController**.

Figure 23.11 Interface para **BNRAssetType**

Crie um novo arquivo template de Objective-C class e escolha `NSObject` como a superclasse. Chame essa classe de **BNRAssetTypeViewController**.

No `BNRAssetTypeViewController.h`, faça a declaração forward de **BNRItem**, altere a superclasse para **UITableViewController** e forneça a ela uma propriedade **BNRItem**.

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

@class BNRItem;

@interface BNRAssetTypeViewController : NSObject
@interface BNRAssetTypeViewController : UITableViewController

@property (nonatomic, strong) BNRItem *item;

@end
```

Esse controlador de visão de tabela mostrará uma lista de tipos de ativos disponíveis. Tocar em um botão na view da **BNRDetailViewController** a exibirá. Implemente os métodos de fonte de dados e importe os arquivos de cabeçalho adequados no `BNRAssetTypeViewController.m`. (Você já viu tudo isso antes.)

```
#import "BNRAssetTypeViewController.h"
#import "BNRItemStore.h"
#import "BNRItem.h"

@implementation BNRAssetTypeViewController

- (instancetype)init
{
    return [super initWithStyle:UITableViewStylePlain];
}
```

```

- (instancetype)initWithStyle:(UITableViewStyle)style
{
    return [self init];
}

- (void)viewDidLoad
{
    [super viewDidLoad];

    [self.tableView registerClass:[UITableViewCell class]
        forCellReuseIdentifier:@"UITableViewCell"];
}

- (NSInteger)tableView:(UITableView *)tableView
 numberOfRowsInSection:(NSInteger)section
{
    return [[[BNRItemStore sharedStore] allAssetTypes] count];
}

- (UITableViewCell *)tableView:(UITableView *)tableView
    cellForRowAtIndexPath:(NSIndexPath *)indexPath
{
    UITableViewCell *cell =
        [tableView dequeueReusableCellWithIdentifier:@"UITableViewCell"
            forIndexPath:indexPath];

    NSArray *allAssets = [[BNRItemStore sharedStore] allAssetTypes];
    NSManagedObject *assetType = allAssets[indexPath.row];

    // Use key-value coding to get the asset type's label
    NSString *assetLabel = [assetType valueForKey:@"label"];
    cell.textLabel.text = assetLabel;

    // Checkmark the one that is currently selected
    if (assetType == self.item.assetType) {
        cell.accessoryType = UITableViewCellAccessoryCheckmark;
    } else {
        cell.accessoryType = UITableViewCellAccessoryNone;
    }

    return cell;
}

- (void)tableView:(UITableView *)tableView
didSelectRowAtIndexPath:(NSIndexPath *)indexPath
{
    UITableViewCell *cell = [tableView cellForRowAtIndexPath:indexPath];
    cell.accessoryType = UITableViewCellAccessoryCheckmark;

    NSArray *allAssets = [[BNRItemStore sharedStore] allAssetTypes];
    NSManagedObject *assetType = allAssets[indexPath.row];
    self.item.assetType = assetType;

    [self.navigationController popViewControllerAnimated:YES];
}

```

@end

No `BNRDetailViewController.xib`, arraste uma `UIBarButtonItem` para a barra de ferramentas. Crie um outlet para esse botão selecionando a barra de ferramentas, depois o novo botão da barra, e também pressione Control arrastando para a extensão de classe de `BNRDetailViewController.m`. Nomeie esse outlet como `assetTypeButton`. Em seguida, crie uma ação a partir desse botão da mesma maneira, arrastando para a seção `@implementation` em vez da extensão de classe e nomeie-a como `showAssetTypePicker`.

O método a seguir e a variável de instância deverão agora ser declarados no `BNRDetailViewController.m`:

```
@property (weak, nonatomic) IBOutlet UIBarButtonItem *assetTypeButton;  
@end  
  
@implementation BNRDetailViewController  
  
// Other methods here  
  
- (IBAction)showAssetTypePicker:(id)sender  
{  
}  
}  
@end
```

Na parte superior do `BNRDetailViewController.m`, importe o cabeçalho para esse novo controlador de visão de tabela.

```
#import "BNRDetailViewController.h"  
#import "BNRAssetTypeViewController.h"
```

Finalize a implementação de `showAssetTypePicker:` no `BNRDetailViewController.m`.

```
- (IBAction)showAssetTypePicker:(id)sender  
{  
    [self.view endEditing:YES];  
  
    BNRAssetTypeViewController *avc = [[BNRAssetTypeViewController alloc] init];  
    avc.item = self.item;  
  
    [self.navigationController pushViewController:avc  
                                    animated:YES];  
}
```

E, finalmente, atualize o título do botão para mostrar o tipo de ativo de uma `BNRItem`. No `BNRDetailViewController.m`, adicione o código a seguir ao `viewWillAppear:`.

```
if (self.itemKey) {  
    // Get image for image key from image cache  
    UIImage *imageToDisplay = [[BNRImageStore sharedStore]  
                                imageForKey:self.itemKey];  
  
    // Use that image to put on the screen in imageView  
    self.imageView.image = imageToDisplay;  
} else {  
    // clear the imageView  
    imageView.image = nil;  
}  
NSString *typeLabel = [self.item.assetType valueForKey:@"label"];  
if (!typeLabel) {  
    typeLabel = @"None";  
}  
  
self.assetTypeButton.title = [NSString stringWithFormat:@"Type: %@", typeLabel];  
  
[self updateFonts];  
}
```

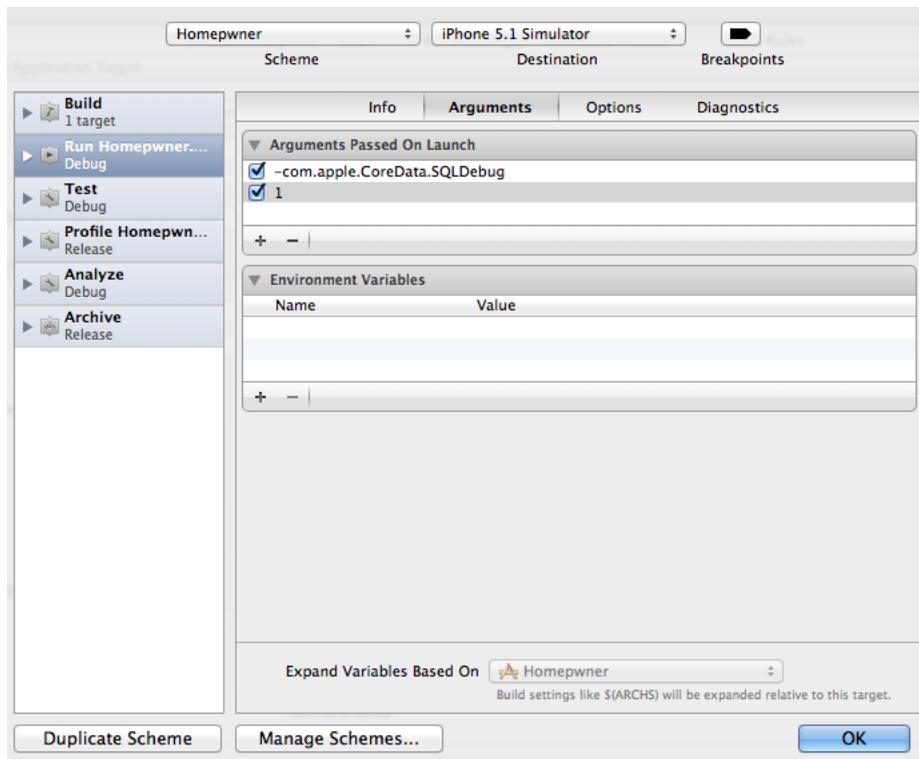
Compile e execute o aplicativo. Selecione uma `BNRItem` e defina seu tipo de ativo.

## Mais sobre SQL

Neste capítulo, você usou SQLite via Core Data. Se estiver curioso à respeito de quais comandos SQL o Core Data está executando, você pode usar um argumento de linha de comando para registrar todas as comunicações com o banco de dados SQLite no console. No menu Product, selecione Edit Scheme.... Selecione o item Run

Homepwner.app e a guia Arguments. Adicione dois argumentos: `-com.apple.CoreData.SQLDebug` e `1`, como mostrado.

Figure 23.12 Ativação do registro do Core Data



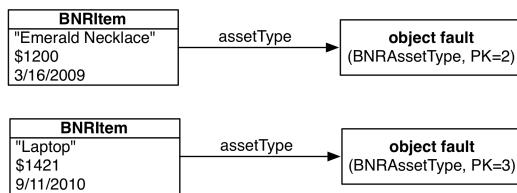
Compile e execute o aplicativo novamente. Certifique-se de que a área de depuração e o console estejam visíveis, para que você possa visualizar o registro de SQL. Adicione algumas localizações e itens de inventário. Em seguida, navegue pelo aplicativo e observe os vários itens.

## Objetos falsos

Os relacionamentos são obtidos de maneira tardia. Quando você busca um objeto gerenciado com relacionamentos, os objetos na outra extremidade desses relacionamentos *não* são obtidos. Em vez disso, o Core Data usa *objetos falsos*. Objetos falsos são objetos placeholders leves que oferecem um ponto final para um relacionamento, até que objetos potencialmente maiores sejam realmente necessários. Isso proporciona benefícios de desempenho e utilização de memória para o gerenciamento de objetos.

Há objetos falsos um para vários (que substituem conjuntos) e um para um (que substituem objetos gerenciados). Então, por exemplo, quando as instâncias de `BNRItem` são obtidas no seu aplicativo, as instâncias de `BNRAssetType` não são. Em vez disso, criam-se objetos falsos que substituem os objetos `BNRAssetType` até que eles sejam realmente necessários.

Figure 23.13 Objetos falsos

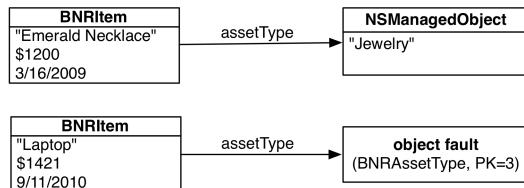


Um objeto falso sabe a qual entidade pertence e qual é sua chave primária. Portanto, por exemplo, quando você perguntar a um objeto falso que representa um tipo de ativo qual é o seu rótulo, você verá um SQL executado que se assemelha ao seguinte:

```
SELECT t0.Z_PK, t0.Z_OPT, t0.ZLABEL FROM ZBNRASSETTYPE t0 WHERE t0.Z_PK = 2
```

(Por que tudo está com o prefixo Z\_? Não sabemos. O que é OPT? Talvez seja uma abreviação de “optimistic locking” (travamento otimista). Esses detalhes não são importantes.) O objeto falso é substituído no mesmo local exato na memória, com um objeto gerenciado contendo os dados reais.

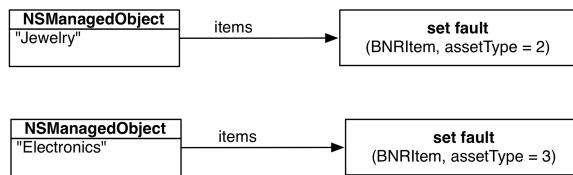
Figure 23.14 Após a substituição de um objeto falso



Essa busca tardia torna o Core Data não só fácil de usar, mas também bastante eficiente.

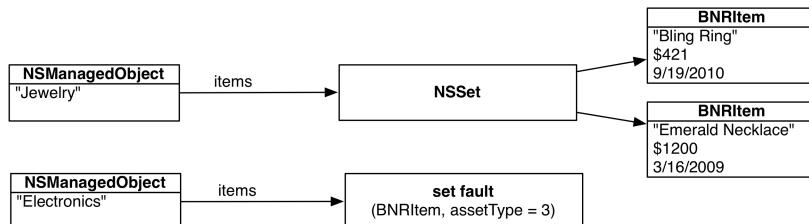
E quanto a objetos falsos um para vários? Vamos supor que seu aplicativo funcionasse de outra maneira: o usuário recebe uma lista de tipos de ativos para selecionar. Em seguida, os itens para esse tipo de ativo são obtidos e exibidos. Como isso funcionaria? Quando os ativos são obtidos pela primeira vez, cada um tem um conjunto falso que substitui a **NSSet** dos objetos do item:

Figure 23.15 Conjuntos falsos



Quando o conjunto falso recebe uma mensagem que exige os objetos **BNRItem**, ele os obtém e substitui a si mesmo por uma **NSSet**:

Figure 23.16 Substituição de conjunto falso



O Core Data é um framework de persistência bastante poderoso e flexível, e este capítulo foi apenas uma rápida introdução a seus recursos. Para obter mais detalhes, sugerimos que leia o *Core Data Programming Guide* (Guia de programação de Core Data) da Apple. Aqui estão algumas coisas que não abordamos:

- A **NSFetchRequest** é um mecanismo poderoso para a especificação de dados que você deseja do armazenamento persistente. Nós o usamos um pouco, mas você vai querer se aprofundar. Você também deve explorar as seguintes classes relacionadas: **NSPredicate**, **NSSortOrdering**, **NSExpressionDescription** e **NSExpression**. Além disso, templates de solicitações de busca podem ser criados como parte do arquivo do modelo.
- Uma *propriedade obtida* parece um pouco com um relacionamento um para vários e com uma **NSFetchRequest**. Geralmente, você as especifica no arquivo do modelo.
- À medida que o seu aplicativo evolui de versão para versão, você precisará alterar o modelo de dados ao longo do tempo. Isso pode ser complicado; na verdade, a Apple tem um livro inteiro sobre isso: *Data Model Versioning and Data Migration Programming Guide*.

- Há um bom suporte para a validação de dados conforme eles entram nas suas instâncias de **NSManagedObject** e, novamente, à medida que passam de seu objeto gerenciado para o armazenamento persistente.
- Você pode ter uma única **NSManagedObjectContext** funcionando com mais de um armazenamento persistente. Você partitiona o seu modelo em *configurações* e, em seguida, atribui cada configuração a um determinado armazenamento persistente. Você não pode ter relacionamentos entre entidades em armazenamentos diferentes, mas pode usar propriedades obtidas para alcançar um resultado semelhante.

## Vantagens e desvantagens de mecanismos persistentes

Neste ponto, você pode começar a se perguntar quais são as vantagens e desvantagens das maneiras comuns que os aplicativos de iOS têm de armazenar dados. Qual é melhor para o seu aplicativo? Use a Table 23.1 para ajudá-lo a decidir.

Table 23.1 Prós e contras do armazenamento de dados

| Técnica      | Prós                                                                                                   | Contras                                                                                                                                                                                                         |
|--------------|--------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Arquivamento | Permite relacionamentos ordenados (arrays, não conjuntos).<br>Fácil de lidar com o controle de versão. | Lê todos os objetos (sem falhas).<br>Sem atualizações incrementais.                                                                                                                                             |
| Serviço web  | Facilita o compartilhamento de dados com outros dispositivos e aplicativos.                            | Requer um servidor e uma conexão com a Internet.                                                                                                                                                                |
| Core Data    | Busca tardia como padrão.<br>Atualizações incrementais.                                                | O controle de versão é estranho (mas certamente pode ser feito usando uma <b>NSModelMapping</b> ).<br>Não há ordenação real dentro de uma entidade, embora relacionamentos um para vários possam ser ordenados. |

## Desafio de bronze: ativos no iPad

No iPad, apresente a **BNRAssetTypeViewController** em uma **UIPopoverController**.

## Desafio de prata: tipos de ativos novos

Torne possível que o usuário adicione tipos de ativos novos, adicionando um botão à `navigationItem` de **BNRAssetTypeViewController**.

## Desafio de ouro: exibição de ativos de um tipo

No controlador de visão da **BNRAssetTypeViewController**, crie uma segunda seção na visão de tabela. Essa seção deve exibir todos os ativos que pertencem ao tipo de ativo selecionado.



# 24

## Restauração de estado

Como discutimos no Chapter 18, os aplicativos têm expectativas de vida limitadas. Se o iOS precisar de mais memória e seu aplicativo estiver em segundo plano, a Apple pode desativá-lo para devolver memória ao sistema. Isso deve ser transparente para seus usuários; eles devem sempre voltar ao último ponto em que estavam no aplicativo.

Para obter essa transparência, você deve adotar a *restauração de estado* em seu aplicativo. A restauração de estado funciona de forma similar a outra tecnologia que você já utilizou para a persistência de dados: o arquivamento. Quando o aplicativo é suspenso, um instantâneo da hierarquia de controlador de visão é salvo. Se o aplicativo foi desativado antes de o usuário abri-lo novamente, o seu estado será restaurado na inicialização. (Se o aplicativo não foi desativado, então está tudo ainda na memória e não haverá necessidade de restaurar qualquer estado.)

Neste capítulo, você adicionará restauração de estado ao aplicativo Homepwner.

Vamos começar demonstrando a necessidade da restauração de estado. Abra o projeto Homepwner. Crie um novo item e busque a sua tela de detalhe (a **BNRDetailViewController**). Agora você precisa simular o processo que aciona a restauração de estado. No iOS Simulator, pressione o botão Home (ou utilize o atalho de teclado Command-Shift-H). Isso colocará o aplicativo em segundo plano. Agora, para desativar o aplicativo como se o sistema o tivesse feito, volte ao Xcode e clique no botão de parar (Command-.). Depois, reinicie o aplicativo a partir do Xcode.

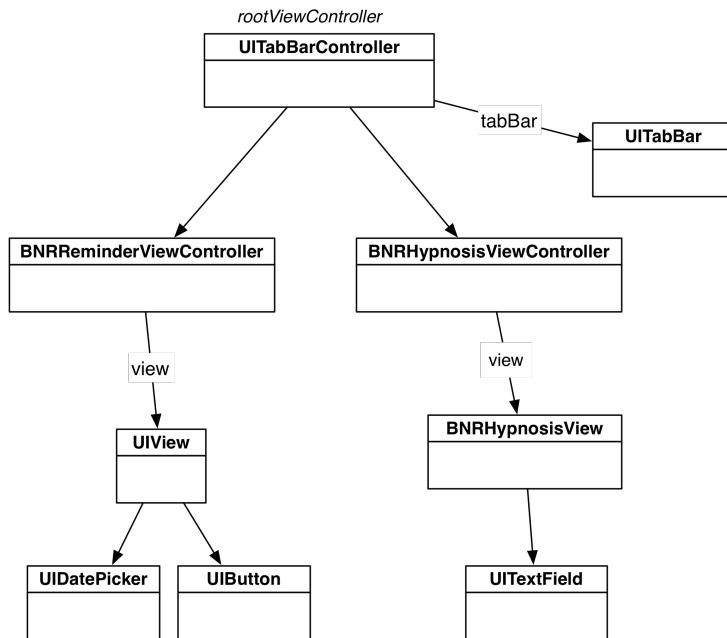
Quando o aplicativo for iniciado, você verá a **BNRDetailViewController** de relance, mas ela será rapidamente substituída pela **BNRItemsViewController**. Por que isso acontece? Quando os aplicativos vão para o segundo plano, o iOS tira um instantâneo da interface do usuário. Depois, quando o aplicativo é reiniciado, esse instantâneo é usado como a imagem de abertura até que o aplicativo seja carregado na memória.

Se um aplicativo não implementar a restauração de estado, o usuário verá rapidamente o estado anterior do aplicativo, mas então a tela mudará para o estado de nova inicialização. No Homepwner, você vê um relance do controlador de visão de detalhes e depois o controlador de visão de itens. Essa é uma experiência confusa.

### Como a restauração de estado funciona

Pode-se pensar em um aplicativo em execução como uma árvore de controladores de visão e visões, com o controlador de visão raiz sendo a raiz da árvore. Por exemplo, pode-se imaginar a interface do seu aplicativo HypnoNerd como uma árvore semelhante à seguinte:

Figure 24.1 Um aplicativo é uma árvore



Se você optar pela restauração de estado, antes de seu aplicativo ser encerrado, o sistema percorrerá essa árvore perguntando a cada nó: “Qual é seu nome?”, “Qual é sua classe?” e “Tem algum dado que você quer que eu grave?”. Enquanto o aplicativo está desativado, essa descrição da árvore fica gravada no sistema de arquivos.

O nome de um nó, que na realidade é conhecido como *identificador de restauração*, normalmente é o nome da classe do objeto. A *classe de restauração* normalmente é a classe do objeto. Os dados retêm o estado do objeto. Por exemplo, os dados de um controlador de visão de guia incluem qual guia está selecionada no momento.

Quando o aplicativo é reiniciado, o sistema tenta recriar a árvore de controladores de visão e as visões dessa descrição salva. Para cada nó:

- O sistema pede à classe de restauração que crie um novo controlador de visão para o nó.
- O novo nó recebe um array de identificadores de restauração: o identificador do nó que está sendo criado e os identificadores de todos os seus ascendentes. O primeiro identificador do array é o identificador do nó raiz. O último é o identificador do nó que está sendo recriado.
- O novo nó recebe seus dados de estado. Esses dados chegam na forma de uma `NSCoder`, que você usou no Chapter 18.

Neste capítulo, você irá estender os controladores de visão do Homeowner para fornecer corretamente sua informação de nó quando o aplicativo está sendo encerrado e para usar essa informação de nó quando eles são recuperados.

## Optando pela restauração de estado

A restauração de estado, por padrão, está desativada nos aplicativos. Para ativá-la, você deve optar por ela no delegate do aplicativo.

Abra o `BNRAppDelegate.m` e implemente os dois métodos delegate para permitir o salvamento e a restauração de estado.

```

@implementation BNRAppDelegate
{
    - (BOOL)application:(UIApplication *)application
        shouldSaveApplicationState:(NSCoder *)coder
    {
        return YES;
    }

    - (BOOL)application:(UIApplication *)application
        shouldRestoreApplicationState:(NSCoder *)coder
    {
        return YES;
    }
}

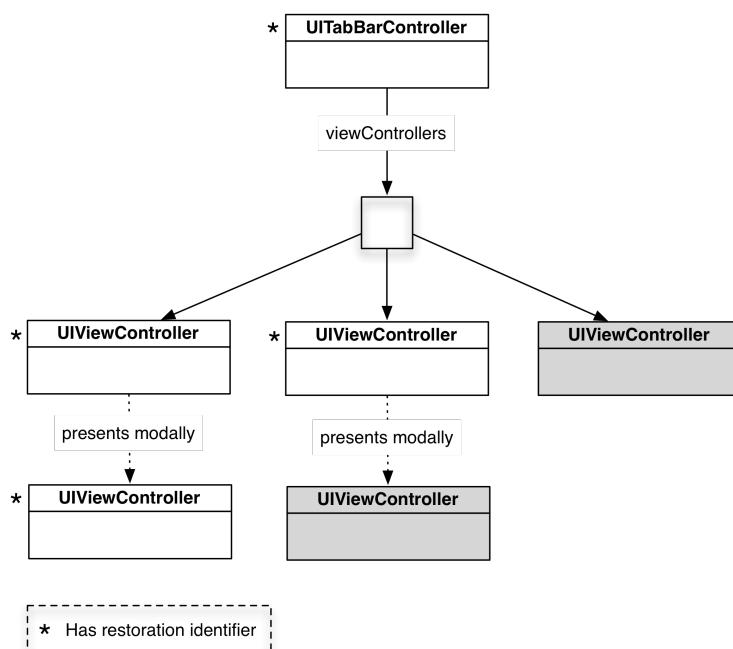
```

Quando o aplicativo for para o segundo plano, seu estado tentará ser salvo e, se o aplicativo estiver sendo iniciado do zero, o seu salvamento tentará ser restaurado. Para entender o que é armazenado, precisamos discutir os *identificadores de restauração*.

## Identificadores e classes de restauração

Quando o estado de um aplicativo está sendo salvo, a `rootViewController` da janela é solicitada a sua `restorationIdentifier`. Se tiver uma `restorationIdentifier`, ela será solicitada a salvar (ou *codificar*) seu estado. Então, ela fica responsável pelo processamento dos controladores de visão filhos da mesma forma, e eles, por sua vez, passam o bastão para *seus* controladores de visão filhos. Entretanto, se qualquer dos controladores de visão na hierarquia não tiver uma `restorationIdentifier`, ela (e seus controladores de visão filhos, tendo ou não `restorationIdentifiers`) será excluída do estado salvo.

Figure 24.2 Identificadores de restauração



Por exemplo, no aplicativo mostrado na Figure 24.2, o estado dos dois controladores de visão cinza, assim como quaisquer controladores de visão filhos que eles possam ter, não seria salvo.

Normalmente, o identificador de restauração de uma classe tem o mesmo nome da própria classe. No `BNRAppDelegate.m`, dê ao controlador de navegação um identificador de restauração no `application:didFinishLaunchingWithOptions:`.

```
- (BOOL)application:(UIApplication *)application
didFinishLaunchingWithOptions:(NSDictionary *)launchOptions
{
    self.window = [[UIWindow alloc] initWithFrame:UIScreen.mainScreen.bounds]];
    // Override point for customization after application launch

    BNRItemsViewController *itemsViewController
        = [[BNRItemsViewController alloc] init];

    // Create an instance of a UINavigationController
    // Its stack contains only itemsViewController
    UINavigationController *navController = [[UINavigationController alloc]
        initWithRootViewController:itemsViewController];

    // Give the navigation controller a restoration identifier that is
    // the same name as the class
    navController.restorationIdentifier = NSStringFromClass([navController class]);

    // Place navigation controller's view in the window hierarchy
    self.window.rootViewController = navController;

    self.window.backgroundColor = [UIColor whiteColor];
    [self.window makeKeyAndVisible];

    return YES;
}
```

Agora que o controlador de navegação tem um identificador de restauração, ele tentará salvar o estado de seus `viewControllers` se eles próprios tiverem identificadores de restauração.

Em suas duas subclasses de `UIViewController` (`BNRItemsViewController` e `BNRDetailViewController`), você atribuirá o identificador de restauração dentro do inicializador designado daquela classe. Além disso, você configurará a *classe de restauração* do controlador de visão. Quando o estado do controlador de visão estiver sendo restaurado, ele pedirá à sua classe de restauração uma instância do controlador de visão a ser restaurado.

Abra o `BNRItemsViewController.m` e atualize o `init` para atribuir o identificador e a classe de restauração.

```
- (instancetype)init
{
    // Call the superclass's designated initializer
    self = [super initWithStyle:UITableViewStylePlain];
    if (self) {
        UINavigationItem *navItem = self.navigationItem;
        navItem.title = @"Homepwner";

        self.restorationIdentifier = NSStringFromClass([self class]);
        self.restorationClass = [self class];
    }
}
```

Abra o `BNRDetailViewController.m` e dê ao novo controlador um identificador e uma classe de restauração no `initForNewItem:`.

```
- (instancetype)initForNewItem:(BOOL)isNew
{
    self = [super initWithNibName:nil bundle:nil];

    if (self) {
        self.restorationIdentifier = NSStringFromClass([self class]);
        self.restorationClass = [self class];

        if (isNew) {
```

Finalmente, há mais um controlador de visão que é criado no Homepwner: `UINavigationController` que é apresentado modalmente quando um novo item é criado.

Reabra o `BNRItemsViewController.m` e atualize o `addNewItem:` para dar a `UINavigationController` um identificador de restauração.

```

- (IBAction)addNewItem:(id)sender
{
    // Create a new BNRItem and add it to the store
    BNRItem *newItem = [[BNRItemStore sharedStore] createItem];

    BNRDetailViewController *detailViewController =
        [[BNRDetailViewController alloc] initForNewItem:YES];

    detailViewController.item = newItem;
    detailViewController.dismissBlock = ^{
        [self.tableView reloadData];
    };

    UINavigationController *navController = [[UINavigationController alloc]
        initWithRootViewController:detailViewController];
    navController.restorationIdentifier = NSStringFromClass([navController class]);
}

```

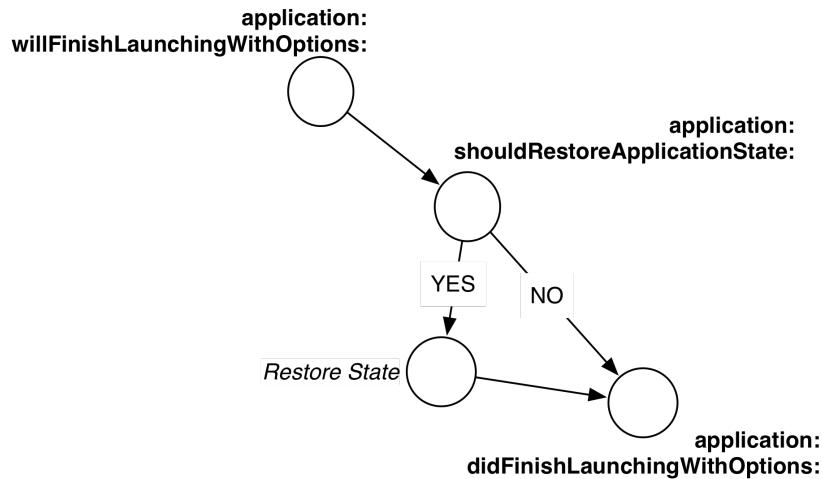
(As duas instâncias de **UINavigationController** que o Homepwner usa não têm classes de restauração. Por causa disso, o delegate do aplicativo receberá a solicitação para criar novas instâncias desses controladores de visão, como você verá em breve.)

Agora que todos os controladores de visão no Homepwner têm um identificador de restauração, o estado deles será salvo quando o usuário deixar o aplicativo.

## Ciclo de vida da restauração de estado

Agora que você está trabalhando com a restauração de estado, o ciclo de vida do aplicativo vai ficar um pouco diferente, como você pode ver na Figure 24.3. Atualmente, todo o seu código de window e rootViewController está no **application:didFinishLaunchingWithOptions:**, mas, com a restauração de estado, ele vai se espalhar um pouco.

Figure 24.3 Ciclo de vida da restauração



O método **application:willFinishLaunchingWithOptions:** é chamado antes que a restauração de estado tenha começado. Você deve utilizar esse método para configurar a window e fazer qualquer coisa que deveria acontecer antes da restauração de estado.

No BNRAAppDelegate.m, sobrescreva esse método para inicializar e configurar a window.

```

- (BOOL)application:(UIApplication *)application
willFinishLaunchingWithOptions:(NSDictionary *)launchOptions
{
    self.window = [[UIWindow alloc] initWithFrame:[[UIScreen mainScreen] bounds]];
    self.window.backgroundColor = [UIColor whiteColor];

    return YES;
}

```

A seguir, atualize o `application:didFinishLaunchingWithOptions:` para configurar a hierarquia do controlador de visão no caso de a restauração de estado não ter ocorrido (por exemplo, na primeira inicialização do aplicativo). Além disso, remova o código que há agora no `application:willFinishLaunchingWithOptions:`.

```
- (BOOL)application:(UIApplication *)application
didFinishLaunchingWithOptions:(NSDictionary *)launchOptions
{
    self.window = [[UIWindow alloc] initWithFrame:UIScreen.mainScreen.bounds]];
    // Override point for customization after application launch.

    // If state restoration did not occur,
    // set up the view controller hierarchy
    if (!self.window.rootViewController) {
        BNRItemsViewController *itemsViewController
            = [[BNRItemsViewController alloc] init];

        // Create an instance of a UINavigationController
        // Its stack contains only itemsViewController
        UINavigationController *navController = [[UINavigationController alloc]
            initWithRootViewController:itemsViewController];

        // Give the navigation controller a restoration identifier that is
        // the same name as the class
        navController.restorationIdentifier = NSStringFromClass([navController class]);

        // Place navigation controller's view in the window hierarchy
        self.window.rootViewController = navController;
    }

    self.window.backgroundColor = [UIColor whiteColor];
    [self.window makeKeyAndVisible];

    return YES;
}
```

## Restauração de controladores de visão

Já que ambos os controladores de visão têm uma classe de restauração, esta será solicitada a criar novas instâncias do seu respectivo controlador de visão. No `BNRItemsViewController.h`, faça com que o controlador de visão fique em conformidade com o protocolo `UIViewControllerRestoration`.

```
@interface BNRItemsViewController : UITableViewController <UIViewControllerRestoration>
@end
```

Depois, no `BNRItemsViewController.m`, implemente o método obrigatório para esse protocolo, que retornará uma nova instância de controlador de visão.

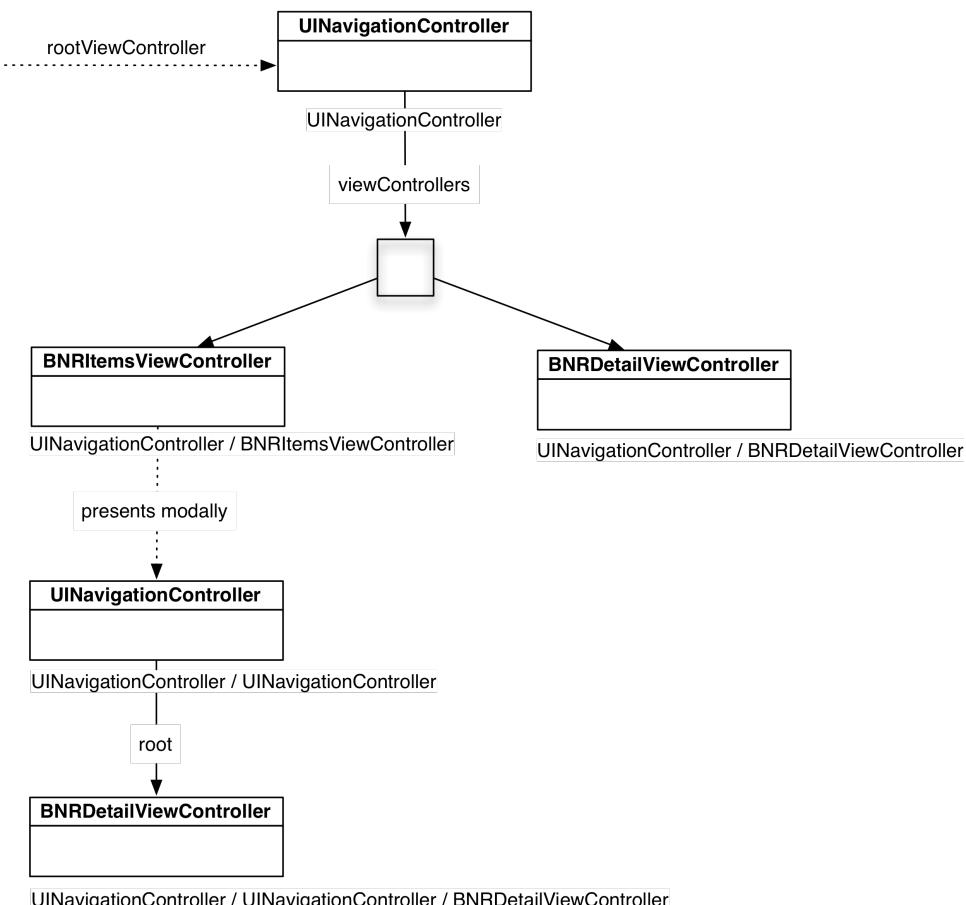
```
+ (UIViewController *)viewControllerWithRestorationIdentifierPath:(NSArray *)path
                                                       coder:(NSCoder *)coder
{
    return [[self alloc] init];
}
```

Agora faça o mesmo para `BNRDetailViewController`: Abra o `BNRDetailViewController.h` e faça com ele fique em conformidade com o protocolo `UIViewControllerRestoration`.

```
@interface BNRDetailViewController : UIViewController <UIViewControllerRestoration>
```

Implementar o método obrigatório do protocolo para `BNRDetailViewController` é um pouco mais difícil, pois você precisa saber se deverá passar YES ou NO para o `initForNewItem:`. Felizmente, o argumento de caminho do identificador de restauração pode ajudar.

Figure 24.4 Caminho de restauração



O identificador de restauração é um array de identificadores de restauração que representa esse controlador de visão e seus ascendentes no momento em que o estado do controlador de visão foi salvo. A Figure 24.4 mostra os caminhos de restauração dos diferentes caminhos de código do aplicativo Homepwner. Um caminho que pode parecer estranho é o caminho de restauração da **UINavigationController** que é apresentada de forma modal. Quando **BNRItemsViewController** apresenta o controlador de navegação de forma modal, na realidade ele é apresentado a partir do pai de **BNRItemsViewController**, que é a **UINavigationController** raiz. Assim, "BNRItemsViewController" não está no caminho do identificador de restauração do controlador de visão apresentado modalmente.

Munido deste conhecimento, você sabe que a count do array de caminho do identificador de restauração será 2 se você estiver vendo uma **BNRItem** existente, e 3 se você estiver criando uma nova **BNRItem**.

Abra o **BNRDetailViewController.m** e implemente o método obrigatório do protocolo **UIViewControllerRestoration**.

```

+ (UIViewController *)viewControllerWithRestorationIdentifierPath:(NSArray *)path
                                                       coder:(NSCoder *)coder
{
    BOOL isNew = NO;
    if ([path count] == 3) {
        isNew = YES;
    }

    return [[self alloc] initForNewItem:isNew];
}
  
```

Agora, **BNRDetailViewController** terá os itens de botão de barra corretos quando o estado for restaurado.

Você cuidou de **BNRItemsViewController** e de **BNRDetailViewController**, mas ainda há dois controladores de visão que deverão ser restaurados: os dois controladores de navegação usados pelo aplicativo. Você deu identificadores de restauração a ambos os controladores de visão, mas você não dará a eles uma classe de restauração.

No lugar disso, se um controlador de visão que estiver sendo restaurado não tiver uma classe de restauração, o delegate do aplicativo será solicitado a fornecer o controlador de visão. Abra o `BNRAppDelegate.m` e implemente esse método.

```
- (UIViewController *)application:(UIApplication *)application
viewControllerWithRestorationIdentifierPath:(NSArray *)identifierComponents
coder:(NSCoder *)coder
{
    // Create a new navigation controller
    UIViewController *vc = [[UINavigationController alloc] init];

    // The last object in the path array is the restoration
    // identifier for this view controller
    vc.restorationIdentifier = [identifierComponents lastObject];

    // If there is only 1 identifier component, then
    // this is the root view controller
    if ([identifierComponents count] == 1) {
        self.window.rootViewController = vc;
    }

    return vc;
}
```

Compile e execute o aplicativo. Crie um item e pesquise-o para ver os detalhes. Acione a restauração de estado assim como você fez no começo deste capítulo. Após reiniciar o aplicativo, você voltará para `BNRDetailViewController`; mas os detalhes do item estarão em branco. Embora a hierarquia do controlador de visão esteja sendo salva no momento, nenhuma informação de modelo é salva, portanto, `BNRDetailViewController` não faz ideia de qual `BNRItem` deve ser exibida. Para corrigir isso, você precisará codificar manualmente uma referência à `BNRItem` que está sendo exibida.

## Codificação de dados relevantes

Para manter outras informações, uma `UIViewController` recebe a chance de codificar quaisquer dados relevantes em um processo bastante similar ao arquivamento. Na verdade, ambos os processos de codificação utilizam um objeto `NSCoder` para fazer o trabalho. Você fará uso disso para salvar quaisquer informações necessárias nas subclasses do controlador de visão.

No `BNRDetailViewController.m`, codifique a `itemKey` para a `BNRItem` atualmente exibida.

```
- (void)encodeRestorableStateWithCoder:(NSCoder *)coder
{
    [coder encodeObject:self.item.itemKey
                 forKey:@"item.itemKey"];

    [super encodeRestorableStateWithCoder:coder];
}
```

Depois, implemente o método de decodificação para procurar a `BNRItem` apropriada na `BNRItemStore`.

```
- (void)decodeRestorableStateWithCoder:(NSCoder *)coder
{
    NSString *itemKey =
        [coder decodeObjectForKey:@"item.itemKey"];

    for (BNRItem *item in [[BNRItemStore sharedStore] allItems]) {
        if ([itemKey isEqualToString:item.itemKey]) {
            self.item = item;
            break;
        }
    }

    [super decodeRestorableStateWithCoder:coder];
}
```

Compile e execute o aplicativo e procure uma `BNRItem`. Depois, execute os passos da restauração de estado novamente. Desta vez, os valores em `BNRDetailViewController` refletirão corretamente a `BNRItem` que está sendo exibida.

Ainda há um problema: os campos de texto e rótulos estão sendo preenchidos com os valores da `BNRItem`. Se o usuário digitou algum outro valor, esses valores serão perdidos na restauração de estado.

Você irá corrigir isso salvando os valores atuais de campo de texto na **BNRItem**. Como a codificação do controlador de visão acontece depois de o aplicativo entrar no segundo plano, você também precisará salvar novamente o armazenamento.

No **BNRDetailViewController.m**, atualize o método de codificação.

```
- (void)encodeRestorableStateWithCoder:(NSCoder *)coder
{
    [coder encodeObject:self.item.itemKey
        forKey:@"item.itemKey"];

    // Save changes into item
    self.item.itemName = self.nameField.text;
    self.item.serialNumber = self.serialNumberField.text;
    self.item.valueInDollars = [self.valueField.text intValue];

    // Have store save changes to disk
    [[BNRItemStore sharedStore] saveChanges];

    [super encodeRestorableStateWithCoder:coder];
}
```

A **BNRDetailViewController** agora está salvando e restaurando o seu estado perfeitamente. Sem muito trabalho de sua parte, seus usuários ganham uma experiência melhor. Agora, vamos nos focar em **BNRItemsViewController**.

## Salvamento de estados de visão

Há vários problemas com **BNRItemsViewController**:

- A **UITableView** não está sendo restaurada para sua posição de rolagem existente, e a linha atualmente selecionada (caso o usuário tenha procurado um item) não está sendo restaurada.
- O controlador de visão não salva a informação de se está ou não no modo de edição e, assim, sempre será restaurado com o valor padrão (que é o de não estar no modo de edição).

Você irá corrigir o segundo item em breve, de forma muito similar à como você codificou informações para **BNRDetailViewController**, mas, antes de fazer isso, vamos dar uma olhada no primeiro item.

Além dos controladores de visão terem um identificador de restauração, certas subclasses **UIView** podem ter um identificador de restauração para salvar determinadas informações sobre a visão. Especificamente, essas subclasses podem salvar algumas de suas informações: **UICollectionView**, **UIImageView**, **UIScrollView**, **UITableView**, **UITextField**, **UITextView** e **UIWebView**.

A documentação de cada uma dessas classes declara quais informações são preservadas. Para a **UITableView**, a informação mais importante salva é o deslocamento do conteúdo da visão de tabela (a posição de rolagem).

No **BNRItemsViewController.m**, dê à **UITableView** um identificador de restauração.

```
- (void)viewDidLoad
{
    [super viewDidLoad];

    // Load the NIB file
    UINib *nib = [UINib nibWithNibName:@"BNRItemCell" bundle:nil];

    // Register this NIB which contains the cell
    [self.tableView registerNib:nib
        forCellReuseIdentifier:@"BNRItemCell"];

    self.tableView.restorationIdentifier = @"BNRItemsViewControllerTableView";
}
```

Compile e execute o aplicativo e role para baixo em **BNRItemsViewController** (você provavelmente terá de adicionar alguns itens novos). Acione a restauração de estado e, na reinicialização, a visão de tabela retornará ao seu deslocamento de conteúdo anterior.

Vamos voltar nossa atenção ao modo de edição do controlador de visão, para que o estado seja mantido. No **BNRItemsViewController.m**, implemente os métodos de codificação e de decodificação.

```
- (void)encodeRestorableStateWithCoder:(NSCoder *)coder
{
    [coder encodeBool:self.isEditing forKey:@"TableViewIsEditing"];
    [super encodeRestorableStateWithCoder:coder];
}

- (void)decodeRestorableStateWithCoder:(NSCoder *)coder
{
    self.editing = [coder decodeBoolForKey:@"TableViewIsEditing"];
    [super decodeRestorableStateWithCoder:coder];
}
```

Você tem um último problema para resolver para concluir a restauração de estado no aplicativo Homepwner. A **UITableView** salvará informações sobre a linha **UITableViewCell** selecionada, mas você terá de facilitar isso um pouco.

No **BNRItemsViewController.m**, faça com que o controlador de visão esteja em conformidade com o protocolo **UIDataSourceModelAssociation** na extensão de classe.

```
@interface BNRItemsViewController : UIViewController
    <UIPopoverControllerDelegate, UIDataSourceModelAssociation>
@property (nonatomic, strong) UIPopoverController *imagePopover;
@end
```

O protocolo **UIDataSourceModelAssociation** ajuda a restauração de estado a localizar os objetos de modelo apropriados para seu aplicativo. Quando o estado do aplicativo estiver sendo salvo, a restauração de estado pedirá um identificador único para o objeto de modelo associado à linha ou às linhas selecionadas (uma **BNRItem**, por exemplo). Quando o aplicativo for reinicializado, a restauração de estado fornecerá o identificador e pedirá um caminho de índice para esse objeto de modelo. Objetos de modelo podem trocar de posição na **UITableView** na reinicialização, mas, desde que seu mapeamento esteja correto, as linhas corretas serão selecionadas.

No **BNRItemsViewController.m**, implemente o método para fornecer à restauração de estado um identificador único da **BNRItem** selecionada. Você utilizará a propriedade **itemKey** como o identificador único.

```
- (NSString *)modelIdentifierForElementAtIndexPath:(NSIndexPath *)path
                                         inView:(UIView *)view
{
    NSString *identifier = nil;

    if (path && view) {
        // Return an identifier of the given NSIndexPath,
        // in case next time the data source changes
        BNRItem *item = [[BNRItemStore sharedStore] allItems][path.row];
        identifier = item.itemKey;
    }

    return identifier;
}
```

Depois, implemente o método inverso, que retorna uma **NSIndexPath** de um determinado identificador.

```
- (NSIndexPath *)indexPathForElementWithModelIdentifier:(NSString *)identifier
                                              inView:(UIView *)view
{
    NSIndexPath *indexPath = nil;

    if (identifier && view) {
        NSArray *items = [[BNRItemStore sharedStore] allItems];
        for (BNRItem *item in items) {
            if ([identifier isEqualToString:item.itemKey]) {
                int row = [items indexOfObjectIdenticalTo:item];
                indexPath = [NSIndexPath indexPathForRow:row inSection:0];
                break;
            }
        }
    }

    return indexPath;
}
```

Compile e execute o aplicativo e acione a restauração de estado. O Homepwner agora está totalmente configurado com restauração de estado e proporcionará aos usuários uma boa experiência.

## Desafio de prata: outro aplicativo

Implemente a restauração de estado no aplicativo HypnoNerd. O processo será praticamente idêntico ao que você fez neste capítulo. Certifique-se de codificar o texto de **UITextField** e a data atualmente selecionada de **UIDatePicker**.

(Dica: para concluir completamente esse desafio, a **BNRHypnosisView** também deverá salvar e restaurar os rótulos que foram adicionados a ela. Você terá de dar à **BNRHypnosisView** um identificador de restauração e implementar os métodos de codificação e decodificação apropriados.)

## Para os mais curiosos: controle de instantâneos

Como mencionamos neste capítulo, o sistema tira um instantâneo do aplicativo quando ele passa para o segundo plano. Às vezes você pode querer ter algum controle sobre o que está sendo exibido para seus usuários na próxima inicialização (que também é o que seus usuários irão ver quando visualizarem seu aplicativo na tela de multitarefas).

Por exemplo, se seu aplicativo exibe informações sigilosas (como um aplicativo bancário que mostra números de contas e balanços), você pode querer ocultar essa informação de olhos não autorizados. Outro exemplo: a Apple desfoca o conteúdo da câmera quando o aplicativo Camera passa para o segundo plano, não por motivos de segurança, mas para que os usuários percebam mais facilmente o aplicativo Camera na tela de multitarefas (em vez de se distraírem com o que a câmera estava mostrando quando o aplicativo passou para o segundo plano).

Modificar o instantâneo é fácil: você atualiza a visão antes de o instantâneo ser tirado. Assim, você faz com que ele mostre o que você quiser.

Falamos no Chapter 18 sobre os vários estados pelo qual o aplicativo passa e implementamos os callbacks de delegate de aplicativo para ver as mudanças de estado. Além dos callbacks de delegate de aplicativo, o aplicativo também envia notificações durante a transição entre os estados. Observar as notificações apropriadas em seus controladores de visão dará a você uma oportunidade de atualizar a interface do usuário.

Especificamente, você irá querer observar **UIApplicationWillResignActiveNotification** obscurecer qualquer coisa que seja necessária, e **UIApplicationDidBecomeActiveNotification** preparar as coisas para que o usuário as veja novamente.

```
NSNotificationCenter *nc = [NSNotificationCenter defaultCenter];
[nc addObserver:self
    selector:@selector(applicationResigningActive:)
    name:UIApplicationWillResignActiveNotification
    object:nil];
[nc addObserver:self
    selector:@selector(applicationBecameActive:)
    name:UIApplicationDidBecomeActiveNotification
    object:nil];

- (void)applicationResigningActive:(NSNotification *)note
{
    // Prepare app for snapshot
}

- (void)applicationBecameActive:(NSNotification *)note
{
    // Undo any changes before user returns to app
}
```

Finalmente, se o seu aplicativo implementa a restauração de estado mas, por algum motivo, não fizer sentido utilizar um instantâneo na próxima vez que ele for iniciado, você pode pedir ao aplicativo para ignorar o instantâneo:

```
// This method should be called during the code that preserves the state
[[UIApplication mainApplication] ignoreSnapshotOnNextApplicationLaunch];
```

Pode fazer sentido, por exemplo, se o seu aplicativo estava exibindo uma mensagem de erro de conexão de rede. O usuário pode muito bem ter uma conexão de rede da próxima vez que iniciar o aplicativo e, assim, para reduzir a confusão do usuário, você ignoraria o instantâneo para a próxima inicialização do aplicativo. Em seu lugar, o aplicativo usaria a imagem de abertura.

# 25

## Localização

O atrativo do iOS é global – usuários de iOS vivem em muitos países diferentes e falam idiomas diferentes. Você pode garantir que o seu aplicativo esteja pronto para este público global através dos processos de internacionalização e localização. *Internacionalização* é garantir que sua informação cultural nativa não esteja codificada permanentemente no aplicativo. (Por informação cultural, queremos dizer idioma, moeda, formato de dados, formato numérico, entre outros.)

*Localização*, por outro lado, é o processo de fornecimento de dados adequados em seu aplicativo, com base nas configurações de Language (idioma) e Region Format (formato da região) do usuário. Você pode encontrar estas configurações no aplicativo **Settings**. Selecione a linha General e, em seguida, a linha International.

Figure 25.1 Configurações internacionais



A Apple torna estes processos relativamente simples. Um aplicativo que aproveita as APIs de localização nem precisa ser recompilado para ser distribuído em outros idiomas ou regiões. Neste capítulo, você vai localizar a visão de detalhes de itens do aplicativo Homepwner. (A propósito, “internacionalização” e “localização” são palavras longas. Algumas vezes, você encontrará as abreviações `i18n` e `l10n`, respectivamente.)

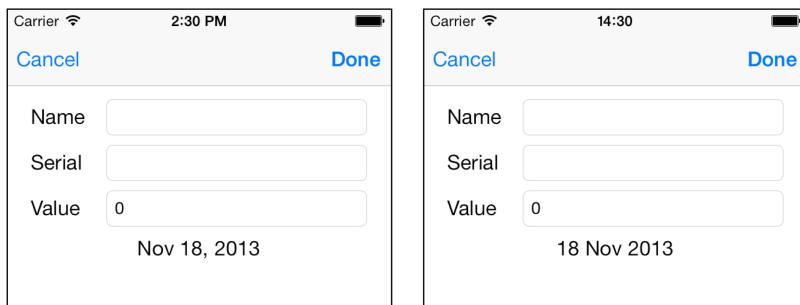
### Internacionalização usando `NSNumberFormat`

Na primeira seção, você usará a classe `NSNumberFormat` para internacionalizar o formato numérico e o símbolo monetário para o valor de um item.

Você sabia que o Homepwner já está parcialmente internacionalizado? Inicie o aplicativo e adicione um novo item. O rótulo de data na `BNRDetailViewController` está formatado de acordo com as configurações regionais atuais. Nos EUA, as datas são mostradas como *Mês Dia, Ano*. Cancele a adição de um novo item.

Agora, abra o aplicativo Settings e altere Region Format para United Kingdom (General → International → Region Format). Volte para o Homepwner e adicione um novo item novamente. Desta vez, a data é exibida como *Dia Mês Ano*. O texto do rótulo de data já foi internacionalizado. Quando isso aconteceu?

Figure 25.2 Formato da data: EUA vs. Reino Unido



No Chapter 10, você usou uma instância de **NSDateFormatter** para definir o texto do rótulo de data de **BNRDetailViewController**. A **NSDateFormatter** tem uma propriedade **locale**, que é definida para a localização atual do dispositivo. Sempre que você usa uma classe **NSDateFormatter** para criar uma data, ela verifica a propriedade **locale** e define o formato adequadamente. Assim, o texto do rótulo de data foi internacionalizado desde o início.

**NSLocale** sabe como regiões diferentes exibem símbolos, datas e decimais e se usam o sistema métrico. Uma instância de **NSLocale** representa as configurações de uma região para estas variáveis. No aplicativo Settings, o usuário pode escolher uma região, como United States ou United Kingdom. (Por que a Apple usa “região” em vez de “país”? Alguns países têm mais de uma região com configurações diferentes. Navegue pelas opções em Region Format para ver por si mesmo.)

Quando você envia a mensagem **currentLocale** para **NSLocale**, a instância de **NSLocale** que representa a configuração da região do usuário é retornada. Uma vez que você tem essa instância de **NSLocale**, pode fazer perguntas como: “Qual é o símbolo monetário para esta região?” ou “Esta região usa o sistema métrico?”.

Para fazer uma dessas perguntas, você envia à instância de **NSLocale** a mensagem **objectForKey:**, com uma das constantes de **NSLocale** como argumento. (Você pode encontrar todas estas constantes na página de documentação sobre **NSLocale**.)

```
NSLocale *locale = [NSLocale currentLocale];
BOOL isMetric = [[locale objectForKey:NSLocaleUsesMetricSystem] boolValue];
NSString *currencySymbol = [locale objectForKey:NSLocaleCurrencySymbol];
```

Vamos internacionalizar o valor exibido em cada **BNRItemCell**. Abra o **Homepwner.xcodeproj**.

No **BNRItemsViewController.m**, adicione uma nova propriedade **currencyFormatter** dentro da extensão de classe.

```
@interface BNRItemsViewController () <UIPopoverControllerDelegate>

@property (nonatomic, strong) UIPopoverController *imagePopover;
@property (nonatomic, strong) NSNumberFormatter *currencyFormatter;
```

@end

Crie **currencyFormatter** no método **init**.

```
[self.navigationItem.leftBarButtonItem = [self editButtonItem]];

NSNotificationCenter *nc = [NSNotificationCenter defaultCenter];
[nc addObserver:self
    selector:@selector(updateTableViewForDynamicTypeSize)
    name:UIContentSizeCategoryDidChangeNotification
    object:nil];

// Create a number formatter for currency
_currencyFormatter = [[NSNumberFormatter alloc] init];
_currencyFormatter.numberStyle = NSNumberFormatterCurrencyStyle;
}

return self;
```

Também no `BNRItemsViewController.m`, localize o método `tableView:cellForRowAtIndexPath:`. Quando o texto da `valueLabel` da célula é definido nesse método, a string `"$%d"` é usada, o que torna o símbolo monetário sempre o sinal de dólar. Substituir esse código com o seguinte exibirá o valor formatado adequadamente para a região do usuário, com o formato numérico e o símbolo monetário.

```
cell.serialNumberLabel.text = item.serialNumber;
cell.valueLabel.text = [NSString stringWithFormat:@"$%d",
    item.valueInDollars];
cell.valueLabel.text = [self.currencyFormatter
    stringFromNumber:@(item.valueInDollars)];

cell.thumbnailView.image = item.thumbnail;

return cell;
}
```

Compile e execute o aplicativo. Você verá o valor formatado de acordo com a região atualmente selecionada, que deve ser United Kingdom se você seguiu as instruções apresentadas no início desta seção. No aplicativo **Settings**, altere Region Format de volta para United States (General → International → Region Format). Retorne para o Homepwner.

Provavelmente, você esperava ver os valores exibidos em dólares (US\$). No entanto, isto não aconteceu. Para ativar a atualização da visão de tabela, comece a adicionar um novo item e cancele imediatamente. Agora, você verá os valores formatados corretamente, pois `viewWillAppear:` foi chamado e recarregou a visão de tabela. (Observe que esta não é uma conversão de moeda; você está apenas substituindo o símbolo.)

Para que o Homepwner se atualize quando as configurações de região mudarem, você precisa usar `NSNotificationCenter`. No método `init` de `BNRItemsViewController`, registre-se para notificações de mudança de localização:

```
NSNotificationCenter *nc = [NSNotificationCenter defaultCenter];
[nc addObserver:self
    selector:@selector(updateTableViewForDynamicTypeSize)
    name:UIContentSizeCategoryDidChangeNotification
    object:nil];

_currencyFormatter = [[NSNumberFormatter alloc] init];
_currencyFormatter.numberStyle = NSNumberFormatterCurrencyStyle;

// Register for locale change notifications
[nc addObserver:self
    selector:@selector(localeChanged:)
    name:NSCurrentLocaleDidChangeNotification
    object:nil];
}

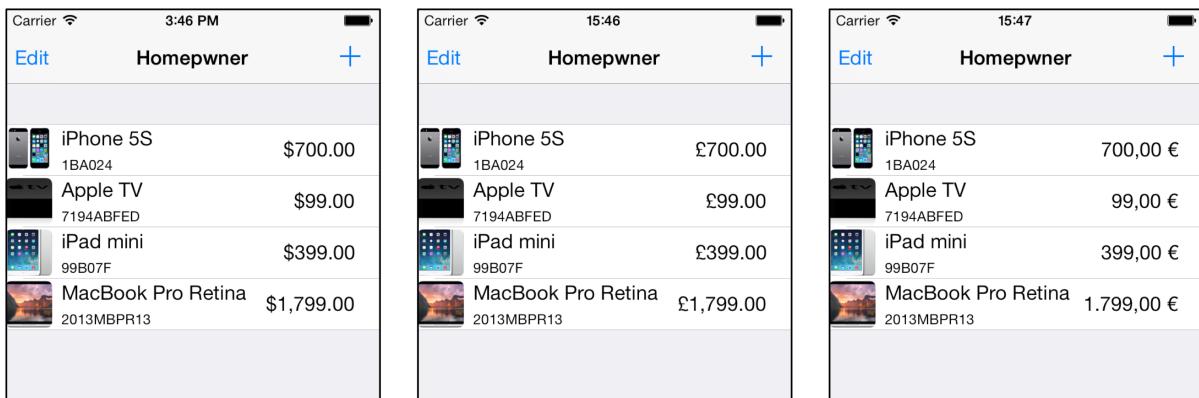
return self;
```

Adicione o método `localeChanged`.

```
- (void)localeChanged:(NSNotification *)note
{
    [self.tableView reloadData];
}
```

Compile e execute o aplicativo. Agora, quando você alterar as configurações regionais e voltar para Homepwner, a visão de tabela será recarregada e o rótulo de valor exibirá o valor devidamente formatado. Para ver por que você usou o formatador numérico em vez de apenas recuperar o símbolo monetário, mude a região para Germany. Não apenas o símbolo monetário mudou, mas algumas outras coisas também: a posição do símbolo monetário (depois do número, em vez de antes), o espaçamento (um espaço entre o valor e o símbolo monetário, em vez de nenhum espaço), o separador decimal (vírgula em vez de ponto) e o separador de milhares (ponto em vez de vírgula).

Figure 25.3 Formato numérico: EUA vs. Reino Unido vs Alemanha



## Recursos de localização

Ao internacionalizar, você faz perguntas à instância de `NSLocale`. Mas a `NSLocale` tem apenas algumas variáveis específicas por região. É aqui que a localização entra em cena: localização é o processo pelo qual as substituições específicas dos aplicativos são criadas para configurações de idiomas e regiões diferentes. Localização, geralmente, significa uma de duas coisas:

- gerar várias cópias de recursos, como imagens, sons e arquivos NIB para regiões e idiomas diferentes
- criar e acessar *tabelas de strings* para traduzir textos em idiomas diferentes

Qualquer recurso, seja uma imagem ou arquivo XIB, pode ser localizado. Localizar um recurso insere outra cópia do recurso no pacote do aplicativo. Estes recursos são organizados em diretórios de idiomas específicos, conhecidos como diretórios `lproj`. Cada um destes diretórios é o nome da localização com o sufixo `lproj`. Por exemplo, a localização do inglês americano é `en_US`: onde `en` é o código para o idioma inglês e `US` é o código para a região dos Estados Unidos da América. (A região pode ser omitida, caso você não precise fazer distinções de região em seus arquivos de recursos.) Estes códigos de idiomas e regiões são padrão em todas as plataformas, não só no iOS.

Quando se pede a um pacote pelo caminho de um arquivo de recurso, ele procura primeiro por um arquivo com este nome no nível raiz do pacote. Se não encontra nenhum, ele procura nas configurações de local e idioma do dispositivo, encontra o diretório `lproj` apropriado e procura ali pelo arquivo. Assim, apenas por localizar os arquivos de recurso, o seu aplicativo carregará o arquivo correto automaticamente.

Uma opção é criar arquivos XIB separados e editar cada string manualmente nesse arquivo XIB no Xcode. No entanto, esta abordagem não é bem dimensionada, se você planeja várias localizações. O que acontece quando você adiciona um novo rótulo ou botão ao XIB localizado? Você precisa adicionar essa visão ao XIB para todos os idiomas. O que não é nada divertido.

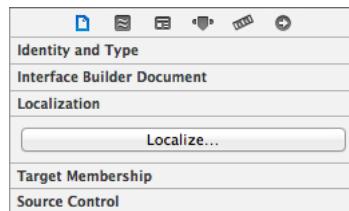
Para simplificar o processo de localização de arquivos XIB, o Xcode tem um recurso chamado Base internationalization (internacionalização de base). Quando ele está ativado para o projeto, o recurso Base internationalization cria o diretório `Base.lproj` que contém os principais arquivos XIB. A localização de arquivos XIB individuais pode ser feita criando apenas os arquivos `Localizable.strings`. Ainda é possível criar todos os arquivos XIB, caso a localização não possa ser feita apenas alterando as strings. No entanto, com a ajuda do Auto Layout, a substituição das strings pode ser suficiente para a maioria das necessidades de localização.

Nesta seção, você vai localizar uma das interfaces do Homepwner: o arquivo `BNRDetailViewController.xib`. Você criará localizações em inglês e espanhol, as quais criará dois diretórios `lproj`, além do diretório de base. Normalmente, primeiro você ativaría o Base Internationalization nas configurações de informações do projeto. Contudo, quando este guia foi escrito, havia um bug no Xcode que não permite a ativação dessa opção antes que pelo menos um arquivo XIB seja localizado.

Portanto, comece com a localização de um arquivo XIB. Selecione o `BNRDetailViewController.xib` no navegador de projetos. Em seguida, mostre a área de utilitários.

Clique na guia no seletor de inspetor para abrir o *inspetor de arquivos*. Encontre a seção chamada Localization nesse inspetor e clique no botão Localize... (Figure 25.4).

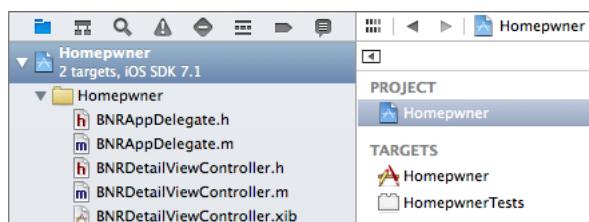
Figure 25.4 Localização de BNRDetailViewController.xib, início



Selecione English. Isso indica ao Xcode que o arquivo pode ser localizado, e automaticamente cria en.lproj e move o arquivo BNRDetailViewController.xib para ele.

Agora você precisa ativar o Base Internationalization (internacionalização de base). Selecione o arquivo do projeto, conforme mostrado na Figure 25.5. Certifique-se de que selecionou o projeto Homepwner e não o destino Homepwner.

Figure 25.5 Seleção de informações do projeto

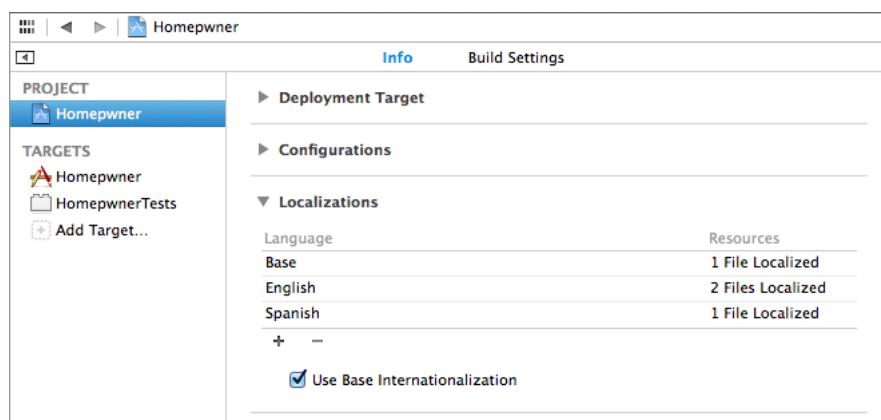


Na seção inferior da guia Info do projeto, localize a caixa de seleção Use Base Internationalization na seção Localizations e selecione-a. Você será solicitado a selecionar quais arquivos serão usados para criar a localização de base; a tabela conterá apenas BNRDetailViewController.xib e o idioma de referência apresentado será o inglês. Clique em Finish.

Isso criará o diretório Base.lproj e moverá BNRDetailViewController.xib para ele.

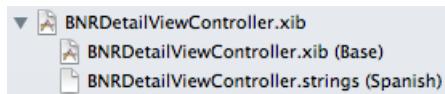
Clique no botão + abaixo da lista de idiomas e selecione Spanish. Na caixa de diálogo, você pode desmarcar os arquivos InfoPlist.strings e manter apenas o arquivo BNRDetailViewController.xib selecionado. Verifique se o idioma de referência é Base e o tipo de arquivo é Localizable Strings. Clique em Finish. Isso cria uma pasta es.lproj e gera o BNRDetailViewController.strings nela, contendo todas as strings do arquivo XIB de base. A configuração de Localizations deve ser semelhante à da Figure 25.6.

Figure 25.6 Localizações



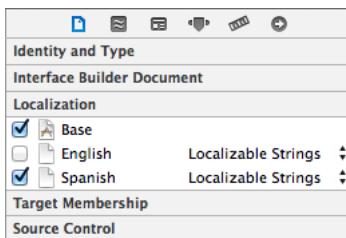
Examine o navegador de projetos. Clique no botão de divulgação ao lado do `BNRDetailViewController.xib` (Figure 25.7). O Xcode moveu o arquivo `BNRDetailViewController.xib` para o diretório `Base.lproj` e criou o arquivo `BNRDetailViewController.strings` no diretório `es.lproj`.

Figure 25.7 XIB localizado no navegador de projetos



Selecione o `BNRDetailViewController.xib`. Não importa se você selecionou aquele no nível superior ou aquele com a marcação Base. O inspetor de arquivos deve se parecer com a Figure 25.8.

Figure 25.8 Localizaçāo de `BNRDetailViewController.xib`, resultado



No navegador de projetos, clique na versão Spanish de `BNRDetailViewController.strings`. Quando o arquivo é aberto, o texto não está em espanhol. Você mesmo precisa traduzir os arquivos localizados. O Xcode não é *tão* inteligente assim.

Edita esse arquivo de acordo com o texto a seguir. Os números e a ordem podem ser diferentes em seu arquivo, mas você pode usar o campo `text` no comentário para coincidir com as traduções.

```
/* Class = "IBUILabel"; text = "Serial"; ObjectID = "JkL-nP-h3R"; */
"JkL-nP-h3R.text" = "Número de serie";

/* Class = "IBUILabel"; text = "Label"; ObjectID = "Q5n-Bc-7IH"; */
"Q5n-Bc-7IH.text" = "Label";

/* Class = "IBUILabel"; text = "Name"; ObjectID = "qzL-Fn-qch"; */
"qzL-Fn-qch.text" = "Nombre";

/* Class = "IBUILabel"; text = "Value"; ObjectID = "rhE-7e-oTE"; */
"rhE-7e-oTE.text" = "Valor";

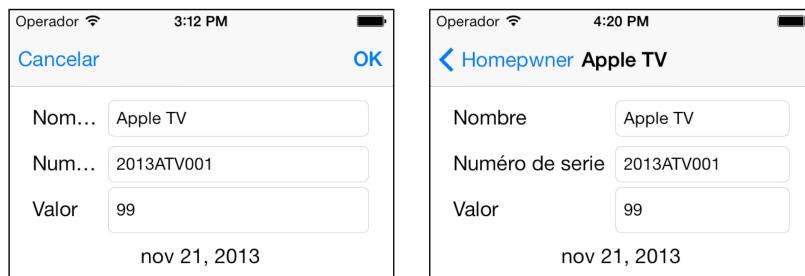
/* Class = "IBUIBarButtonItem"; title = "Item"; ObjectID = "uNg-wM-Zcr"; */
"uNg-wM-Zcr.title" = "Item";
```

Observe que você não alterou o texto `Label` (rótulo) e `Item`, pois essas strings serão substituídas programmaticamente no tempo de execução. Salve esse arquivo.

Agora que você terminou de localizar este arquivo XIB, vamos testá-lo. Primeiro, há uma pequena falha no Xcode da qual precisamos estar cientes: às vezes o Xcode simplesmente ignora as alterações de um arquivo de recurso quando você constrói um aplicativo. Para garantir que o aplicativo esteja sendo construído do zero, primeiro exclua-o do seu dispositivo ou simulador. (Pressione e segure seu ícone no inicializador. Quando ele começar a tremer, toque no emblema de exclusão.) Reinicie o Xcode (sim, saia e inicie novamente). Em seguida, escolha Clean no menu Product. Por fim, para ter certeza absoluta, pressione e segure o botão alt/option ao abrir o menu Product e selecione Clean Build Folder.... Isso forçará o aplicativo a ser recompilado, reagrupado e reinstalado completamente.

A visão de detalhes do Homepwner não aparecerá em espanhol até que você mude as configurações de idioma no dispositivo ou simulador. No Settings, altere as configurações de idioma para Español (General → International → Language) e, em seguida, reinicie o aplicativo. Selecione a linha de um item e você verá a interface em espanhol. Entretanto, os rótulos estão sendo cortados (Figure 25.9).

Figure 25.9 BNRDetailViewController.xib em espanhol, antes e depois da correção do layout



Felizmente, graças ao Auto Layout, é muito fácil corrigir isso. Abra o arquivo `BNRDetailViewController.xib` e selecione o rótulo de nome. Na área de utilitários, selecione o inspetor de tamanho (a guia ) e localize a restrição de largura. Atualmente, a largura está definida para ser igual a 55, o que é muito curto para os rótulos mais longos em espanhol. Clique na lista suspensa ao lado do botão de engrenagem da restrição de largura e escolha `Select and Edit...`. Mude a relação de `Equal` para `Greater Than or Equal`.

Agora, você ajustará as larguras dos rótulos de número de série e valor. Selecione o rótulo de número de série e localize sua restrição de largura. Em vez de editá-la, apenas exclua-a. Você quer que os campos de texto sejam do mesmo tamanho e a maneira de conseguir isso é fazer com que os rótulos tenham a mesma largura. Depois de excluir a restrição de largura, clique no rótulo de número de série com a tecla control pressionada e arraste-o até o rótulo de nome. No menu, selecione o item `Equal widths`. Repita os mesmos passos para o rótulo de valor. Compile e execute o aplicativo. Desta vez, os campos de texto serão redimensionados para dar espaço aos rótulos, de forma que eles não sejam cortados.

## NSMutableString e tabelas de strings

Em muitos lugares do seu aplicativo, você cria instâncias de `NSString` de forma dinâmica ou exibe strings literais para o usuário. Para exibir versões traduzidas destas strings, você deve criar uma *tabela de strings*. A tabela de strings é um arquivo que contém uma lista de pares chave-valor para todas as strings que o aplicativo usa e suas traduções relacionadas. É um arquivo de recurso que você adiciona ao aplicativo, mas você não precisa de muito trabalho para obter dados dele.

Você pode usar uma string em seu código semelhante à seguinte:

```
NSString *greeting = @"Hello!"
```

Para internacionalizar a string em seu código, você substitui as strings literais com a função `NSMutableString`.

```
NSString *greeting = NSLocalizedString(@"Hello!", @"The greeting for the user");
```

Essa função leva dois argumentos: uma chave e um comentário que descreve o uso da string. A chave é o valor de pesquisa em uma tabela de strings. No tempo de execução, `NSMutableString()` examinará as tabelas de strings que acompanham seu aplicativo, em busca de uma tabela que corresponda às configurações de idioma do usuário. Em seguida, nessa tabela, a função obtém a string traduzida que corresponde à chave.

Agora você vai internacionalizar a string “Homepwner” que é exibida na barra de navegação. No `BNRItemsViewController.m`, encontre o método `init` e mude a linha de código que define o título da `navigationItem`.

```
- (instancetype)init
{
    // Call the superclass's designated initializer
    self = [super initWithStyle:UITableViewStylePlain];
    if (self) {
        UINavigationItem *navItem = [self navigationItem];
        navItem.title = @"Homepwner";
        navItem.title = NSLocalizedString(@"Homepwner", @"Name of application");
    }
}
```

Outros dois controladores de visão contêm strings hard-coded (embutidas no código) que podem ser internacionalizadas. A barra de ferramentas no **BNRDetailViewController** mostra o tipo do ativo. O título de **BNRAssetTypeViewController** precisa ser atualizado exatamente como o título de **BNRItemsViewController**.

No **BNRDetailViewController.m**, atualize o método **viewWillAppear::**:

```
NSString *typeLabel = [self.item.assetType valueForKey:@"label"];
if (!typeLabel) {
    typeLabel = @"None";
    typeLabel = NSLocalizedString(@"None", @"Type label None");
}
self.assetTypeButton.title = [NSString stringWithFormat:@"Type: %@", typeLabel];
self.assetTypeButton.title = [NSString stringWithFormat:
    NSLocalizedString(@"Type: %@", @"Asset type button"), typeLabel];

[self updateFonts];
}
```

No **BNRAssetTypeViewController.m**, atualize o método **init**:

```
if (self) {
    self.navigationItem.title = @"Asset Type";
    self.navigationItem.title =
        NSLocalizedString(@"Asset Type", @"BNRAssetTypeViewController title");
}
return self;
}
```

Uma vez que você tenha arquivos que foram internacionalizados com a função **NSLocalizedString**, você pode gerar tabelas de strings com um aplicativo de linha de comando.

Abra o aplicativo Terminal. Caso não o tenha usado antes, este é um terminal Unix; ele é utilizado para executar ferramentas de linha de comando. Você quer navegar para o local de **BNRItemsViewController.m**. Se você nunca usou o aplicativo Terminal antes, aqui vai um truque útil. No Terminal, digite o seguinte:

cd

seguido de um espaço. (Ainda não pressione Enter.)

Em seguida, abra Finder e localize **BNRItemsViewController.m** e a pasta onde ele está contido. Arraste o ícone dessa pasta para a janela do Terminal. O Terminal preencherá o caminho para você. Pressione Enter.

O diretório de trabalho atual de Terminal agora é este. Por exemplo, meu comando de terminal será semelhante ao seguinte:

```
cd /Users/aaron/Homepwner/Homepwner/
```

Use o comando terminal **ls** para imprimir o conteúdo do diretório de trabalho e confirme que **BNRItemsViewController.m** se encontra na lista.

Para gerar a tabela de strings, insira o seguinte no Terminal e pressione a tecla Enter:

```
genstrings BNRItemsViewController.m
```

Isso cria um arquivo chamado **Localizable.strings** no mesmo diretório que **BNRItemsViewController.m**. Agora você precisa gerar as strings a partir dos outros dois controladores de visão. Como o arquivo **Localizable.strings** já existe, você vai querer anexar a ele, em vez de criá-lo do zero. Para tanto, insira os seguintes comandos no Terminal (não se esqueça da opção de linha de comando **-a**) e pressione Enter depois de cada linha:

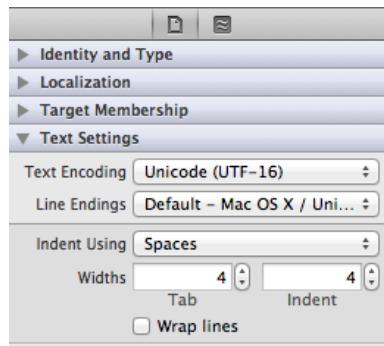
```
genstrings -a BNRDetailViewController.m
genstrings -a BNRAssetTypeViewController.m
```

O arquivo resultante **Localizable.strings** agora contém as strings dos três controladores de visão. Arraste-o do Finder para o navegador de projetos (ou utilize o item de menu **Add Files to "Homepwner"...**). Quando o aplicativo for compilado, esse recurso será copiado para o pacote principal.

Por mais estranho que pareça, o Xcode às vezes apresenta um problema com tabelas de strings. Abra o arquivo **Localizable.strings** na área do editor. Caso veja uma série de pontos de interrogação invertidos, você precisa

reinterpretar este arquivo como Unicode (UTF-16). Exiba a área de utilitários e selecione o inspetor de arquivos. Encontre a área chamada **Text Settings** e altere o menu pop-up ao lado de **Text Encoding** para **Unicode (UTF-16)** (Figure 25.10). Você será questionado se deseja reinterpretar ou converter. Escolha **Reinterpret**.

Figure 25.10 Alteração da codificação de um arquivo



O arquivo deve ser semelhante ao seguinte:

```
/* Name of application */
"Homepwner" = "Homepwner";

/* BNRAAssetTypeViewController title */
"Asset Type" = "Asset Type";

/* Type label None */
"None" = "None";

/* Asset type button */
"Type: %@", "Type: %@";
```

Observe que o comentário acima da sua string é o segundo argumento que você forneceu à função **NSLocalizedString**. Embora a função não exija o argumento de comentário, incluí-lo tornará a localização muito mais fácil.

Agora que você criou **Localizable.strings**, localize-o no Xcode da mesma forma que você fez com o arquivo XIB. Selecione o arquivo no navegador de projetos e clique no botão **Localize...** na área de utilitários. Adicione a localização em espanhol e, em seguida, abra a versão em espanhol de **Localizable.strings**. A string do lado esquerdo é a *chave* que é passada para a função **NSLocalizedString** e a string do lado direito é o que é retornado. Mude o texto do lado direito para a tradução em espanhol apresentada abaixo. (Para digitar ñ, pressione Option + n e, em seguida, “n”.)

```
/* Name of application */
"Homepwner" = "Dueño de casa"

/* AssetTypePickerController title */
"Asset Type" = "Tipo de activo";

/* Type label None */
"None" = "Nada";

/* Asset type button */
"Type: %@", "Tipo: %@";
```

Compile e execute o aplicativo novamente. Agora, todas essas strings, inclusive o título da barra de navegação, aparecerão em espanhol. Caso não apareçam, você pode precisar excluir o aplicativo, limpar o projeto e recompilar. (Ou verifique a configuração de idioma do usuário.)

## Desafio de bronze: outra localização

A prática leva à perfeição. Localize **Homepwner** para outro idioma. Use o Google Translate se precisar de ajuda com o idioma.

## Para os mais curiosos: o papel de **NSBundle** na internacionalização

O verdadeiro trabalho de adicionar uma localização é feito para você pela classe **NSBundle**. Por exemplo, quando uma **UIViewController** é inicializada, ela recebe dois argumentos: o nome de um arquivo XIB e um objeto de **NSBundle**. O argumento do pacote normalmente é **nil**, que é interpretado como o *pacote principal* do aplicativo. (O pacote principal é outro nome para o pacote do aplicativo – todos os recursos e o arquivo executável do aplicativo. Quando um aplicativo é compilado, todos os diretórios **lproj** são copiados para este pacote.)

Quando o controlador carrega sua visão, pergunta ao pacote pelo arquivo XIB. O pacote, sendo muito inteligente, verifica a configuração atual de idioma do dispositivo e procura no diretório **lproj** apropriado. O caminho para o arquivo XIB no diretório **lproj** é retornado para o controlador de visão e carregado.

**NSBundle** sabe como procurar em diretórios de localização por todos os tipos de recursos, usando o método de instância **pathForResource:ofType:**. Quando você quer o caminho para um recurso oferecido com o seu aplicativo, envia esta mensagem ao pacote principal. Veja um exemplo que usa o arquivo de recurso **myImage.png**:

```
NSString *path = [[NSBundle mainBundle] pathForResource:@"myImage"
                                              ofType:@".png"];
```

O pacote verifica primeiro para ver se há um arquivo **myImage.png** no nível superior do pacote do aplicativo. Se encontrado, ele retorna o caminho completo para o arquivo. Caso contrário, o pacote pega a configuração de idioma do dispositivo e procura no diretório **lproj** adequado para construir o caminho. Se nenhum arquivo for encontrado, ele retorna **nil**.

É por isso que você deve excluir e limpar o aplicativo quando localiza um arquivo. O arquivo anterior não localizado ainda estará no nível raiz do pacote do aplicativo, pois o Xcode não excluirá um arquivo do pacote quando você reinstalá-lo. Embora haja pastas **lproj** no pacote do aplicativo, o pacote encontra o arquivo do nível superior e retorna o seu caminho.

## Para os mais curiosos: localização de arquivos XIB sem o Base Internationalization

Antes que o Xcode tivesse o recurso Base Internationalization, a opção de strings localizáveis não estava disponível. Em vez disso, você manteria um arquivo XIB para todas as localizações para as quais quisesse suporte. Portanto, você teria em **en.lproj/BNRDetailViewController.xib** e em **es.lproj/BNRDetailViewController.xib**. Como você pode imaginar, manter cada XIB em cada idioma para o qual quisesse suporte seria muito complicado.

Para ajudar na criação e manutenção de arquivos XIB localizados, você poderia usar uma ferramenta de linha de comando chamada **ibtool** para extrair as strings do arquivo XIB de seu idioma nativo para um arquivo de strings. Depois, você traduziria essas strings e criaria um novo arquivo XIB para cada idioma.

Para experimentar, abra Terminal e navegue para o diretório **en.lproj**. Por exemplo, meu comando de terminal será semelhante ao seguinte:

```
cd /Users/aaron/Homepwner/Homepwner/en.lproj
```

Em seguida, use o **ibtool** para extrair as strings desse arquivo XIB. Insira o seguinte comando de terminal (tudo na mesma linha) e tecle Enter. (Nós só o quebramos para caber na página.)

```
ibtool --export-strings-file ~/Desktop/BNRDetailViewController.strings
      BNRDetailViewController.xib
```

Isso criará um arquivo **BNRDetailViewController.strings** na sua área de trabalho que contém todas as strings do arquivo XIB. Abra o **BNRDetailViewController.strings** em espanhol. Este é o mesmo arquivo que o Xcode criou utilizando as strings localizáveis. Edite esse arquivo como você fez antes.

Agora você usará **ibtool** para criar um novo arquivo XIB em espanhol. Esse arquivo será baseado na versão em inglês de **BNRDetailViewController.xib**, mas substituirá todas as strings com os valores de

BNRDetailViewController.strings. Para conseguir isso, você precisa saber o caminho do arquivo XIB em inglês e o caminho do diretório em espanhol no diretório do projeto. Lembre-se, você abriu estas janelas no Finder anteriormente.

No Terminal.app, insira o comando abaixo, seguido de espaço depois de write. (Mas ainda não pressione Enter!)

```
ibtool --import-strings-file ~/Desktop/BNRDetailViewController.strings --write
```

Em seguida, encontre BNRDetailViewController.xib em es.lproj e arraste-o para a janela do terminal.

Depois, encontre BNRDetailViewController.xib na pasta en.lproj e arraste-o para esta mesma janela. O seu comando deve ser semelhante ao seguinte:

```
ibtool --import-strings-file ~/Desktop/BNRDetailViewController.strings --write  
/iphone/Homepwner/Homepwner/es.lproj/BNRDetailViewController.xib  
/iphone/Homepwner/Homepwner/en.lproj/BNRDetailViewController.xib
```

Esse comando diz: “Crie BNRDetailViewController.xib no es.lproj a partir de BNRDetailViewController.xib no en.lproj e, em seguida, substitua todas as strings com os valores de BNRDetailViewController.strings.”

Pressione Enter. (Você pode ver algum tipo de aviso em que ibtool reclama sobre GSCapabilities. Pode ignorá-lo.)

Abra o BNRDetailViewController.xib (Spanish) no Xcode. Esse arquivo XIB agora está localizado para o espanhol. Para finalizar, redimensione os campos label (rótulo) e text (texto) para o número de série, conforme mostrado na Figure 25.11.

Figure 25.11 BNRDetailViewController.xib em espanhol





# 26

## NSUserDefaults

Quando você inicia um aplicativo pela primeira vez, ele usa as configurações de fábrica. Bons aplicativos memorizam as suas preferências conforme são usados. Onde as suas preferências são gravadas? Dentro de cada pacote de aplicativo há uma *plist* que armazena as preferências do usuário. Como desenvolvedor, você acessará essa plist utilizando a classe **NSUserDefaults**. A plist de preferências do seu aplicativo também pode ser editada pelo aplicativo **Settings**. Para que isso aconteça, você cria um *pacote de configurações* dentro do seu aplicativo.

Neste capítulo, você ensinará o Homepwner a ler e gravar as preferências do usuário. Então você criará um pacote de configurações.

### NSUserDefaults

O conjunto de configurações padrão de um usuário é uma coleção de pares chave-valor. A chave é o nome do padrão e o valor é algum dado que representa a preferência do usuário para aquela chave. Você pergunta o valor da chave para o objeto compartilhado de padrões do usuário, mais ou menos como obter um objeto de um dicionário:

```
NSUserDefaults *defaults = [NSUserDefaults standardUserDefaults];
NSString *greeting = [defaults objectForKey:@"FavoriteGreeting"];
```

Se o usuário expressar uma preferência, você pode definir o valor para aquela chave:

```
NSUserDefaults *defaults = [NSUserDefaults standardUserDefaults];
[defaults setObject:@"Hello" forKey:@"FavoriteGreeting"];
```

O valor será armazenado automaticamente na plist de preferências do aplicativo. Assim, o valor deve ser um tipo plist: **NSArray**, **NSDictionary**, **NSString**, **NSData**, **NSDate** ou **NSNumber**. Se você quiser gravar um tipo não plist nos padrões do usuário, você deverá convertê-lo para uma plist. Frequentemente, isso é feito arquivando-se o objeto (ou os objetos) em uma **NSData**, que é uma plist.

E se você perguntar pelo valor de uma preferência que não foi definida pelo usuário? **NSUserDefaults** retornará as configurações de fábrica, o “padrão padrão”, se preferir. Elas não são armazenadas no sistema de arquivos, de forma que você deverá informar à instância compartilhada de **NSUserDefaults** quais são as configurações de fábrica toda vez que o aplicativo for iniciado. E você precisa fazer isso logo no começo do processo de inicialização, antes que alguma de suas classes tente ler os padrões. Tipicamente, você irá sobrescrever **+initialize** no seu delegate de aplicativo:

```
+ (void)initialize
{
    NSUserDefaults *defaults = [NSUserDefaults standardUserDefaults];
    NSDictionary *factorySettings = @{@"FavoriteGreeting": @"Hey!",
                                       @"HoursBetweenMothershipConnection" : @2};
    [defaults registerDefaults:factorySettings];
}
```

O método de classe **initialize** é chamado automaticamente pelo tempo de execução do Objective-C antes da criação da primeira instância dessa classe.

Nesta seção, você adicionará as preferências de valor inicial e nome de um item.

## Registro das configurações de fábrica

No momento da inicialização, a primeira coisa que acontecerá será o registro das configurações de fábrica. Considera-se elegante declarar suas chaves de preferência como constantes globais. Abra o `BNRAppDelegate.h` e declare duas variáveis globais constantes:

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

extern NSString * const BNRNextItemValuePrefsKey;
extern NSString * const BNRNextItemNamePrefsKey;

@interface BNRAppDelegate : UIResponder <UIApplicationDelegate>

@property (strong, nonatomic) UIWindow *window;

@end
```

No `BNRAppDelegate.m`, defina essas variáveis globais e use-as para registrar os padrões de fábrica em `+initialize`:

```
#import "BNRAppDelegate.h"
#import "BNRItemsViewController.h"
#import "BNRItemStore.h"

NSString * const BNRNextItemValuePrefsKey = @"NextItemValue";
NSString * const BNRNextItemNamePrefsKey = @"NextItemName";

@implementation BNRAppDelegate

+ (void)initialize
{
    NSUserDefaults *defaults = [NSUserDefaults standardUserDefaults];
    NSDictionary *factorySettings = @{@"BNRNextItemValuePrefsKey": @75,
                                       BNRNextItemNamePrefsKey: @"Coffee Cup"};
    [defaults registerDefaults:factorySettings];
}
```

## Leitura de uma preferência

Quando você cria um novo item em `BNRItemStore.m`, utilize os valores padrão. Certifique-se de importar `BNRAppDelegate.h` no topo de `BNRItemStore.m` para que o compilador tenha conhecimento de `BNRNextItemValuePrefsKey`.

```
- (BNRItem *)createItem
{
    double order;
    if (_allItems.count == 0) {
        order = 1.0;
    } else {
        order = [[self.privateItems lastObject] orderingValue] + 1.0;
    }

    BNRItem *item = [NSEntityDescription insertNewObjectForEntityForName:@"BNRItem"
                                         inManagedObjectContext:self.context];
    item.orderingValue = order;

    NSUserDefaults *defaults = [NSUserDefaults standardUserDefaults];
    item.valueInDollars = [defaults integerForKey:BNRNextItemValuePrefsKey];
    item.itemName = [defaults objectForKey:BNRNextItemNamePrefsKey];

    // Just for fun, list out all the defaults
    NSLog(@"%@", [defaults dictionaryRepresentation]);

    [self.privateItems addObject:item];
}

return item;
}
```

Observe o método `integerForKey:`. Está lá por conveniência. Isso é equivalente a:

```
item.valueInDollars = [[defaults objectForKey:BNRNextItemValuePrefsKey] intValue];
```

Também há métodos de conveniência para definir e obter valores `float`, `double`, `BOOL` e `NSURL`.

## Alteração de uma preferência

Você poderia criar um controlador de visão para editar essas preferências. Ou, então, poderia criar um pacote de configurações para definir essas preferências. Ou, ainda, poderia tentar adivinhar as preferências do usuário a partir de suas ações. Por exemplo, se o usuário define o valor de um item como \$ 100, esse pode ser uma boa indicação de que o próximo item também possa ser \$ 100. Para este exercício, é isso o que você fará.

Abra o `BNRDetailViewController.m` e edite o método `viewWillDisappear:`.

```
- (void)viewWillDisappear:(BOOL)animated
{
    [super viewWillDisappear:animated];
    [self.view endEditing:YES];

    BNRItem *item = self.item;
    item.itemName = self.nameField.text;
    item.serialNumber = self.serialNumberField.text;

    int newValue = [self.valueField.text intValue];

    // Is it changed?
    if (newValue != item.valueInDollars) {

        // Put it in the item
        item.valueInDollars = newValue;

        // Store it as the default value for the next item
        NSUserDefaults *defaults = [NSUserDefaults standardUserDefaults];
        [defaults setInteger:newValue
                      forKey:BNRNextItemValuePrefsKey];
    }
    item.valueInDollars = [self.valueField.text intValue];
}
```

Importe o `BNRAppDelegate.h` para que o compilador tome conhecimento da constante `BNRNextItemValuePrefsKey`. Compile e execute o seu aplicativo. Crie um item chamado “Xícara de Café” com um valor de \$ 75. O próximo item que você criar deverá ter o mesmo valor como padrão.

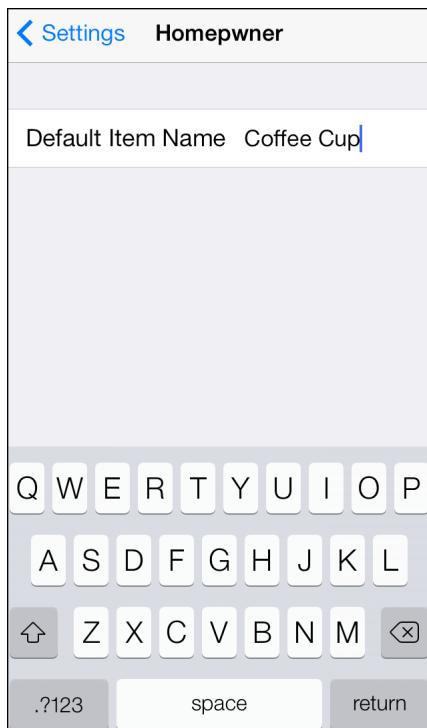
Além disso, no console, você verá uma lista de todos os padrões disponíveis para você. A maioria deles é de `NSGlobalDomain`, que armazena os padrões globais de todo o seu dispositivo, como, por exemplo, que idioma você prefere. O `NextItemName`, como você nunca definiu o valor, está sendo lido a partir dos padrões de fábrica, que são conhecidos como o `NSRegistrationDomain`. Agora que você definiu o `NextItemValue`, ele está sendo lido do domínio `com.bignerdranch.Homepwner`, que é armazenado na plist de preferências na área restrita do seu aplicativo. Você pode pensar em cada domínio como um dicionário de pares chave-valor. A `NSUserDefaults` dá aos dicionários uma precedência diferente. Por exemplo, o domínio `com.bignerdranch.Homepwner` tem precedência sobre o `NSRegistrationDomain` – se o padrão tiver um valor na plist de preferências do aplicativo, o domínio de registro será ignorado.

Algumas vezes você dará ao usuário um botão que diga “Restaurar os padrões de fábrica”, que removerá alguns padrões da plist de preferências do aplicativo. Para remover pares chave-valor da plist de preferências de seu aplicativo, a `NSUserDefaults` tem um método `removeObjectForKey:`.

## Pacote de configurações

Agora você criará um pacote de configurações para que a preferência `NextItemName` possa ser alterada de “Xícara de Café” para qualquer string que o usuário queira.

Figure 26.1 Pacote de configurações do Homepwner

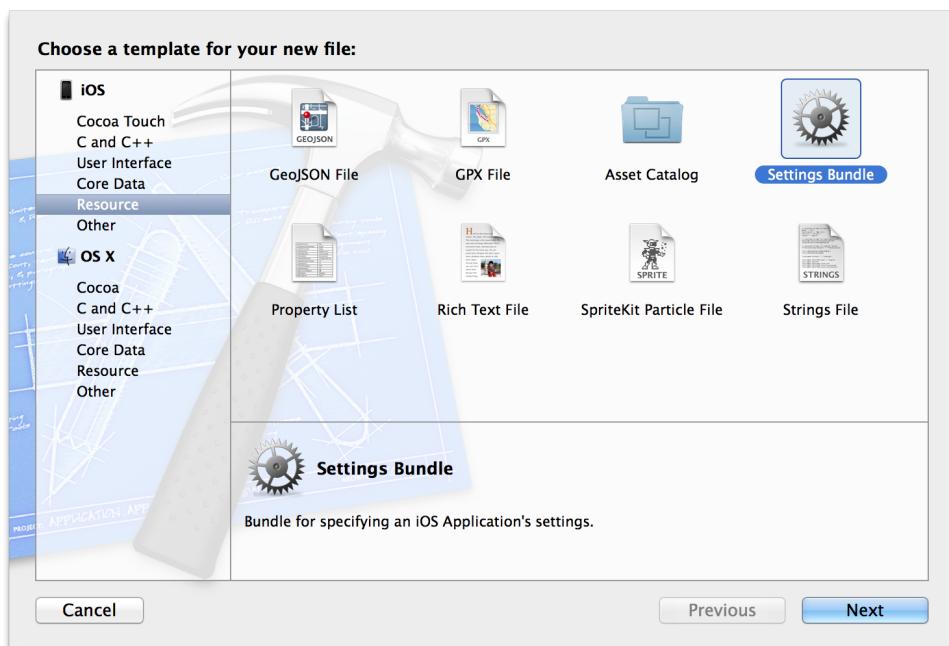


Nos dias de hoje, muitos programadores consideram os pacotes de configurações de mau gosto e a maioria dos aplicativos não inclui tais pacotes. Dito isso, muitos aplicativos *têm* pacotes de configurações; assim, é bom saber como criá-los.

A expressão “pacote de configurações” faz a coisa soar mais assustadora do que realmente é. O pacote é apenas um diretório que armazena uma plist que descreve quais controles deveriam aparecer nessa visão e qual padrão cada controle está manipulando. Você colocará as strings visíveis do usuário (como o rótulo “Default Item Name” (Nome padrão do item) na Figure 26.1) dentro de um arquivo de strings localizado para o usuário. Esses arquivos de strings também estarão no pacote de configurações.

Para criar um pacote de configurações dentro de seu aplicativo, abra o menu File do Xcode e escolha New → File.... No painel iOS Resources, selecione Settings Bundle (Figure 26.2).

Figure 26.2 Criação de um novo pacote de configurações



Aceite o nome do padrão. Perceba que um diretório chamado `Settings.bundle` foi criado no diretório de seu projeto. Possui um arquivo `Root.plist` e um subdiretório `en.lproj`.

## Edição da Root.plist

O `Root.plist` descreve que controles aparecerão no painel de configurações do seu aplicativo. Contém um array de dicionários; cada dicionário representa uma visão (normalmente um controle) que aparecerá no painel. Cada dicionário deverá ter a chave `Type`. Os valores aceitáveis de `Type` são os seguintes:

|                                      |                                                                        |
|--------------------------------------|------------------------------------------------------------------------|
| <code>PSTextFieldSpecifier</code>    | um campo de texto rotulado                                             |
| <code>PSToggleSwitchSpecifier</code> | um botão de alternância rotulado                                       |
| <code>PSSliderSpecifier</code>       | uma barra deslizante (não rotulada)                                    |
| <code>PSRadioGroupSpecifier</code>   | uma lista de botões de opção; apenas um pode ser selecionado           |
| <code>PSMultiValueSpecifier</code>   | uma visão de tabela de possibilidades; apenas uma pode ser selecionada |
| <code>PSTitleValueSpecifier</code>   | um título para formatação                                              |
| <code>PSGroupSpecifier</code>        | um grupo para formatação                                               |
| <code>PSChildPaneSpecifier</code>    | permite que você move algumas preferências para um painel subordinado  |

Vários desses aparecem no `Root.plist` padrão. Examine bem e então compile e execute o `Homepwner`. Assim que o `Homepwner` estiver executando, vá para o aplicativo `Settings` e observe o painel do `Homepwner`.

Voltando para o Xcode, abra o `Root.plist` e reduza-a para um array contendo apenas um campo de texto:

- Defina o `Identifier` como `NextItemName`. Esta é a chave para o padrão que está sendo definido.
- Defina o `DefaultValue` como `Coffee Cup`. É isso o que aparece se não houver um valor para `Key` na plist de preferências do seu aplicativo.
- Defina o `Title` como `NextItemName`. Isso é usado para procurar o título no arquivo de strings.

Suas configurações devem estar parecidas com Figure 26.3.

Figure 26.3 Root.plist

| Key                      | Type       | Value          |
|--------------------------|------------|----------------|
| ▼ iPhone Settings Schema | Dictionary | (2 items)      |
| ▼ Preference Items       | Array      | (1 item)       |
| ▼ Item 0 (Text Field -   | Dictionary | (7 items)      |
| Default Value            | String     | Coffee Cup     |
| Text Field Is Secure     | Boolean    | NO             |
| Identifier               | String     | NextItemName   |
| Keyboard Type            | String     | Alphabet       |
| Title                    | String     | NextItemName   |
| Type                     | String     | Text Field     |
| Autocorrection Style     | String     | Autocorrection |
| Strings Filename         | String     | Root           |

Perceba que você está planejando uma interface do usuário utilizando um arquivo plist. Quando você cria um pacote de configurações, não está escrevendo nenhum código executável nem criando quaisquer controladores de visão ou outros objetos que você controla. O aplicativo *Settings* lerá o *Root.plist* do seu aplicativo e construirá seus próprios controladores de visão com base no conteúdo do arquivo plist.

Se você está criando um pacote de configurações para um de seus aplicativos, você precisará consultar a *Settings Application Schema Reference* da Apple para obter uma lista completa de todas as chaves/valores que funcionarão nessa plist.

## Root.strings localizadas

Dentro de seu pacote de configurações, há um *en.lproj* que armazenará as strings em inglês. Você pode excluir todos os pares chave-valor e dar o título para o seu campo de texto:

```
"NextItemName" = "Default Item Name";
```

Tudo pronto. Compile e execute seu aplicativo. Ele deverá usar o nome de item do aplicativo *Settings* quando você criar um novo item.

Uma última observação: quando seus padrões forem alterados (por seu próprio aplicativo ou pelo aplicativo *Settings*) uma *NSUserDefaultsDidChangeNotification* será enviada para seu aplicativo. Se você quiser reagir imediatamente a mudanças no aplicativo *Settings*, registre-se como observador dessa notificação.

# 27

# Controle de animações

A palavra “animação” vem do latim e significa “ação de dar vida”. Animação é o que dá vida a seus aplicativos e, quando usada de forma adequada, pode guiar os usuários ao longo de diversas ações, orientá-los e, em geral, proporcionar uma experiência prazerosa.

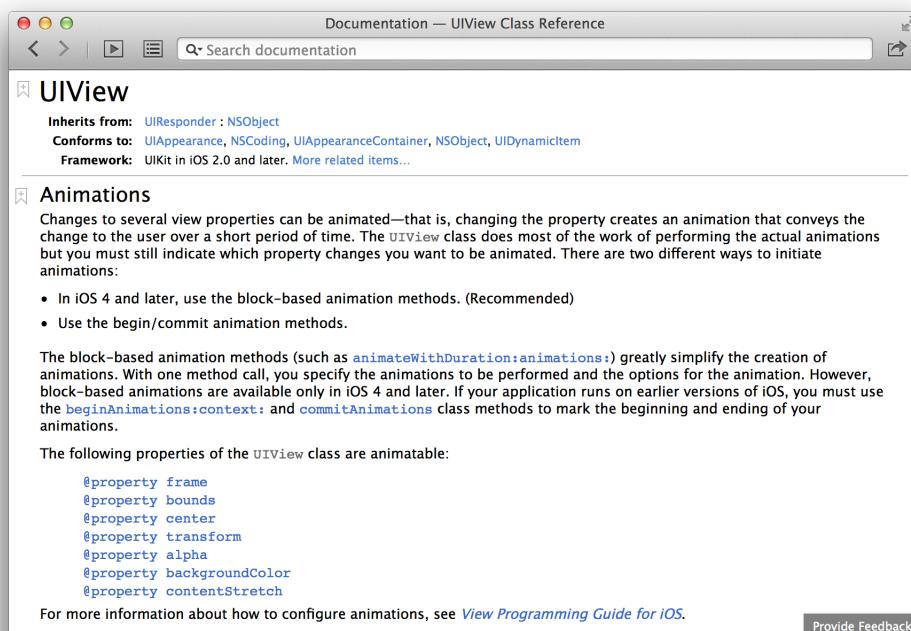
Neste capítulo, você usará diversas técnicas de animação para animar várias visões no aplicativo HypnoNerd.

## Animações básicas

As animações são uma ótima maneira de adicionar uma camada extra de refinamento a qualquer aplicativo; os jogos não são o único tipo de aplicativo a se beneficiar das animações. As animações podem trazer suavemente elementos de interface à tela ou colocá-los em foco, elas chamam a atenção do usuário para um item que permite ação e oferecem informações claras sobre a resposta do seu aplicativo às ações do usuário.

Antes de atualizar o aplicativo HypnoNerd, vamos dar uma olhada no que é possível animar. Abra a documentação em *UIView Class Reference* (Referência de classe UIView) e role a tela até a seção intitulada *Animations* (Animações). A documentação fornecerá algumas recomendações para animação (as quais seguiremos neste livro) e também listará as propriedades de **UIView** que podem ser animadas (Figure 27.1).

Figure 27.1 Documentação de animação da **UIView**



A documentação é sempre um bom ponto de partida no aprendizado de qualquer tecnologia de iOS. Com a compreensão dessas poucas informações, vamos seguir em frente e colocar algumas animações no HypnoNerd.

O primeiro tipo de animação que você usará é a *animação básica*. A animação básica é feita entre um valor inicial e um valor final (Figure 27.2).

Figure 27.2 Animação básica



Abra HypnoNerd.xcodeproj.

A primeira animação que você adicionará animará o valor `alpha` dos rótulos assim que forem adicionados à visão.

Abra `BNRHypnosisViewController.m` e adicione uma animação aos rótulos de `drawHypnoticMessage:`.

```
[self.view addSubview:nameLabel];

// Set the label's initial alpha
nameLabel.alpha = 0.0;

// Animate the alpha to 1.0
[UIView animateWithDuration:0.5 animations:^{
    nameLabel.alpha = 1.0;
}];

UIInterpolatingMotionEffect *motionEffect =
    [[UIInterpolatingMotionEffect alloc] initWithKeyPath:@"center.x"
        type:UIInterpolatingMotionEffectTypeTiltAlongHorizontalAxis];
```

Compile e execute o aplicativo. Após inserir algum texto e tocar na tecla de retorno, os rótulos terão uma perda gradual de intensidade na visão. As animações oferecem uma experiência de usuário mais agradável do que fazer as visões surgirem do nada.

O método `animateWithDuration:animations:` retorna imediatamente. Ou seja, dá início à animação, mas não aguarda a sua conclusão.

O método mais simples de animação baseada em bloco em `UIView` é `animateWithDuration:animations:`. Esse método inclui a duração que a animação deverá ter e um bloco de alterações a ser animado. A animação seguirá uma curva de animação com início suave/conclusão suave, que fará a animação começar lentamente, acelerar na metade da ação e finalmente desacelerar no final.

## Funções de tempo

A aceleração da animação é controlada por sua função de tempo. O método `animateWithDuration:animations:` usa uma função de tempo de início suave/conclusão suave. Usando uma analogia de condução de um carro, o condutor está com o carro parado e acelera suavemente a uma velocidade constante, então desacelera gradualmente até parar novamente.

Outras funções de tempo são a função linear (velocidade constante do início ao fim), início suave (aceleração em velocidade constante e parada brusca) e conclusão suave (início em velocidade total e desaceleração no final).

Para usar uma dessas outras funções de tempo, você precisará usar o método de animação do `UIView` que permite que as opções sejam especificadas: `animateWithDuration:delay:options:animations:completion:`. Esse método concede controle máximo da animação. Além do bloco de duração e animação, também é possível especificar o atraso antes do início das animações, algumas opções (que veremos em breve), e um bloco de conclusão que será chamado quando a sequência de animação for concluída.

No `BNRHypnosisViewController.m`, altere a animação em `drawHypnoticMessage:` para usar este novo método de animação:

```

[self.view addSubview:nameLabel];

// Set the label's initial alpha
nameLabel.alpha = 0.0;

// Animate the alpha to 1.0
[UIView animateWithDuration:0.3 animations:^{
    nameLabel.alpha = 1.0;
}];

[UIView animateWithDuration:0.5
                     delay:0.0
                   options:UIViewAnimationOptionCurveEaseIn
     animations:^{
        nameLabel.alpha = 1.0;
    }
    completion:NULL];
}

UIInterpolatingMotionEffect *motionEffect =
[[UIInterpolatingMotionEffect alloc] initWithKeyPath:@"center.x"
                                             type:UIInterpolatingMotionEffectTypeTiltAlongHorizontalAxis];

```

Agora, em comparação com o uso da curva de animação padrão com início suave/conclusão suave, a animação será apenas de início suave. O argumento das opções é uma bitmask (máscara de bits), para que você aplique o operador binário OR em diversos valores unidos. Abaixo, há algumas opções úteis que podem ser usadas:

#### Opções de curva de animação

Controlam a aceleração da animação. Os possíveis valores são `UIViewControllerAnimatedOptionCurveEaseInOut`, `UIViewControllerAnimatedOptionCurveEaseIn`, `UIViewControllerAnimatedOptionCurveEaseOut` e `UIViewControllerAnimatedOptionCurveLinear`.

#### `UIViewControllerAnimatedOptionAllowUserInteraction`

Por padrão, não é possível interagir com as visões durante a animação. A especificação dessa opção sobrescreverá o padrão. Essa opção é útil para animações repetitivas, como uma visão de pulsação.

#### `UIViewControllerAnimatedOptionRepeat`

Essa opção repetirá a animação indefinidamente. Ela costuma ser pareada com a opção `UIViewControllerAnimatedOptionAutoreverse`.

#### `UIViewControllerAnimatedOptionAutoreverse`

Esta opção avançará a animação e depois a retrocederá, fazendo a visão retornar ao seu estado inicial.

Não deixe de consultar a seção *Constantes* de *UIView Class Reference* para ver todas as opções possíveis. Abordaremos mais algumas posteriormente neste capítulo.

## Animação de quadro-chave

As animações que você adicionou até aqui foram básicas; elas são feitas de determinado valor até um outro valor. Se você quiser animar as propriedades de uma visão através de mais de dois valores, você utilizará uma *animação de quadro-chave*. Uma animação de quadro-chave pode consistir em qualquer quantidade de quadros-chave individuais (Figure 27.3). Você pode pensar nas animações de quadro-chave como múltiplas animações básicas consecutivas.

Figure 27.3 Animação de quadro-chave



As animações de quadro-chave são definidas de forma similar às animações básicas, porém cada quadro-chave é adicionado separadamente. Para criar uma animação de quadro-chave, use o método de classe **animateKeyframesWithDuration:delay:options:animations:completion:** em **UIView** e adicione quadros-chave no bloco de animação usando o método de classe **addKeyframeWithRelativeStartTime:relativeDuration:animations:**.

No **BNRHypnosisViewController.m**, atualize **drawHypnoticMessage:** para animar o centro dos rótulos primeiro até o meio da tela e depois até outra posição aleatória na tela.

```
[UIView animateWithDuration:0.5
    delay:0.0
    options:UIViewAnimationOptionCurveEaseIn
    animations:^{
        nameLabel.alpha = 1.0;
    }
    completion:NULL];

[UIView animateKeyframesWithDuration:1.0 delay:0.0 options:0 animations:^{
    [UIView addKeyframeWithRelativeStartTime:0 relativeDuration:0.8 animations:^{
        nameLabel.center = self.view.center;
    }];
    [UIView addKeyframeWithRelativeStartTime:0.8 relativeDuration:0.2 animations:^{
        int x = arc4random() % width;
        int y = arc4random() % height;
        nameLabel.center = CGPointMake(x, y);
    }];
} completion:NULL];

UIInterpolatingMotionEffect *motionEffect =
[[UIInterpolatingMotionEffect alloc] initWithKeyPath:@"center.x"
    type:UIInterpolatingMotionEffectTypeTiltAlongHorizontalAxis];
```

As animações de quadro-chave são criadas por meio de **animateKeyframesWithDuration:delay:options:animations:completion:**. Os parâmetros são todos os mesmos da animação básica, exceto que as opções são do tipo **UIViewKeyframeAnimationOptions** em vez de **UIViewAnimationOptions**. A duração deste método equivale à duração de toda a animação.

Os quadros-chave individuais são adicionados por meio de **addKeyframeWithRelativeStartTime:relativeDuration:animations:**. O primeiro argumento é o tempo inicial relativo, que será um valor entre 0 e 1. O segundo argumento é a duração relativa, que é a porcentagem da duração total, e também será um valor entre 0 e 1. O primeiro quadro-chave começa em 0% na animação (tempo inicial relativo de 0,0) e a duração será de 80% do total (duração relativa de 0,8). O último quadro-chave começa em 80% da duração total (tempo inicial relativo de 0,8) e terá duração de 20% do total (duração relativa de 0,2).

Compile e execute o aplicativo e insira algum texto. Os rótulos serão então animados até o centro da tela antes de explodirem em uma posição final aleatória.

## Conclusão da animação

Geralmente, é recomendável saber quando a animação termina. Por exemplo, você pode querer encadear diferentes tipos de animações ou atualizar outro objeto quando uma animação é concluída. Para saber quando a animação termina, passe um bloco para o argumento de conclusão.

Atualize **BNRHypnosisViewController.m** para que uma mensagem seja registrada no console quando a animação for concluída.

```
[UIView animateWithDuration:1.0 delay:0.0 options:0 animations:^{
    [UIView addKeyframeWithRelativeStartTime:0 relativeDuration:0.8 animations:^{
        nameLabel.center = self.view.center;
    }];
    [UIView addKeyframeWithRelativeStartTime:0.8 relativeDuration:0.2 animations:^{
        int x = arc4random() % width;
        int y = arc4random() % height;
        nameLabel.center = CGPointMake(x, y);
    }];
} completion:^(BOOL finished) {
    NSLog(@"Animation finished");
}];

[UIView animateWithDuration:1.0 delay:0.0 options:0 animations:^{
    [UIView addKeyframeWithRelativeStartTime:0 relativeDuration:0.8 animations:^{
        nameLabel.center = self.view.center;
    }];
    [UIView addKeyframeWithRelativeStartTime:0.8 relativeDuration:0.2 animations:^{
        int x = arc4random() % width;
        int y = arc4random() % height;
        nameLabel.center = CGPointMake(x, y);
    }];
} completion:^(BOOL finished) {
    NSLog(@"Animation finished");
}];

UIInterpolatingMotionEffect *motionEffect =
[[UIInterpolatingMotionEffect alloc] initWithKeyPath:@"center.x"
type:UIInterpolatingMotionEffectTypeTiltAlongHorizontalAxis];
```

Compile e execute o aplicativo; as mensagens de registro aparecerão no console assim que a animação for concluída.

Você pode estar se perguntando: “E se a animação se repetir? O bloco de conclusão será executado após cada repetição?” Não, o bloco de conclusão será executado apenas uma vez, bem no final.

## Animações de mola

O iOS tem um mecanismo físico poderoso embutido no SDK, e uma das maneiras mais fáceis de usá-lo é com as novas *animações de mola*. Esse tipo de animação possui uma função de tempo parecida com a de uma mola real. Você o utilizará para animar o campo de texto que cai do topo da tela, como se ele estivesse com uma mola.

No `BNRHypnosisViewController.m`, adicione uma propriedade para o campo de texto à extensão da classe e atualize `loadView` para armazenar a referência no campo de texto. Em seguida, comece com um offscreen para o campo de texto:

```
@interface BNRHypnosisViewController () <UITextFieldDelegate>
@property (nonatomic, weak) UITextField *textField;
@end

@implementation BNRHypnosisViewController
// Other methods

- (void)loadView
{
    BNRHypnosisView *backgroundView =
        [[BNRHypnosisView alloc] initWithFrame:[[UIScreen mainScreen] bounds]];
    UITextField *textField =
        [[UITextField alloc] initWithFrame:CGRectMake(40, 70, 240, 30)];

    self.textField =
        [[UITextField alloc] initWithFrame:CGRectMake(40, -20, 240, 30)];

    // Setting the border style on the text field will allow us to see it easier
    textField.borderStyle = UITextBorderStyleRoundedRect;
    [backgroundView addSubview:textField];

    self.view = backgroundView;
}
@end
```

É melhor dar início à animação assim que a visão aparecer na tela; dessa forma, o código da animação vai para `viewDidAppear:`. Atualmente, não existe nenhuma propriedade apontando para o campo de texto, mas você precisará de uma para atualizar o frame em `viewDidAppear:`.

Agora, no `BNRHypnosisViewController.m`, sobrescreva `viewDidAppear:` para fazer o campo de texto cair usando uma animação de mola.

```
- (void)viewDidAppear:(BOOL)animated
{
    [super viewDidAppear:animated];

    [UIView animateWithDuration:2.0
                          delay:0.0
                        usingSpringWithDamping:0.25
                        initialSpringVelocity:0.0
                            options:0
                        animations:^{
                            CGRect frame = CGRectMake(40, 70, 240, 30);
                            self.textField.frame = frame;
                        }
                        completion:NULL];
}
```

Os componentes individuais deste método são relativamente diretos:

|                              |                                                                                                                      |
|------------------------------|----------------------------------------------------------------------------------------------------------------------|
| <i>duração</i>               | O tempo total que a animação deve durar.                                                                             |
| <i>atraso</i>                | Quanto tempo será aguardado para a animação começar.                                                                 |
| <i>amortecimento da mola</i> | Um número entre 0 e 1. Quando mais próximo de 0, mais oscilação haverá na animação.                                  |
| <i>velocidade da mola</i>    | A velocidade relativa da visão quando a animação está para começar. Você provavelmente sempre escolherá 0 para isso. |
| <i>opções</i>                | <code>UIViewControllerAnimatedOptions</code> , exatamente como com outras animações.                                 |
| <i>animações</i>             | Um bloco de alterações que anima uma ou mais visões.                                                                 |
| <i>conclusão</i>             | Um bloco que será executado quando a animação for concluída.                                                         |

Compile e execute o aplicativo; o campo de texto será animado desde a parte superior da tela, ricocheteando como uma mola no final.

## Desafio de prata: Quiz aprimorado

Adicione algumas animações ao aplicativo Quiz com o qual trabalhou no Chapter 1.

Quando uma nova pergunta ou resposta for exibida, ela deverá aparecer flutuando até o lado esquerdo da tela, e sua opacidade deverá ser animada de 0 para 1 no trajeto. A pergunta ou resposta anterior deverá sair flutuando pelo lado direito da tela enquanto perde sua opacidade.

Brinque com os tempos e as curvas de animação para ficar legal.



# 28

## UIStoryboard

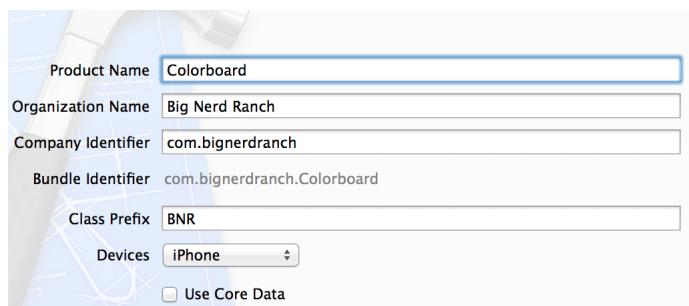
Até o momento, nos seus projetos você distribuiu as interfaces dos seus controladores de visão em arquivos XIB separados e depois instanciou os controladores de visão programaticamente. Neste capítulo, você usará um *storyboard* em vez disso. Storyboards são recursos do iOS que permitem que você instancie e distribua todos os seus controladores de visão em um único arquivo semelhante a um XIB. Além disso, você pode conectar controladores de visão no storyboard para determinar como eles serão apresentados ao usuário.

A finalidade de um storyboard é minimizar um pouco do código mais simples que o programador precisa escrever para criar e configurar controladores de visão e as interações entre eles. Para ver essa simplificação, e as desvantagens dela, vamos usar um aplicativo que usa um storyboard.

### Criação de um storyboard

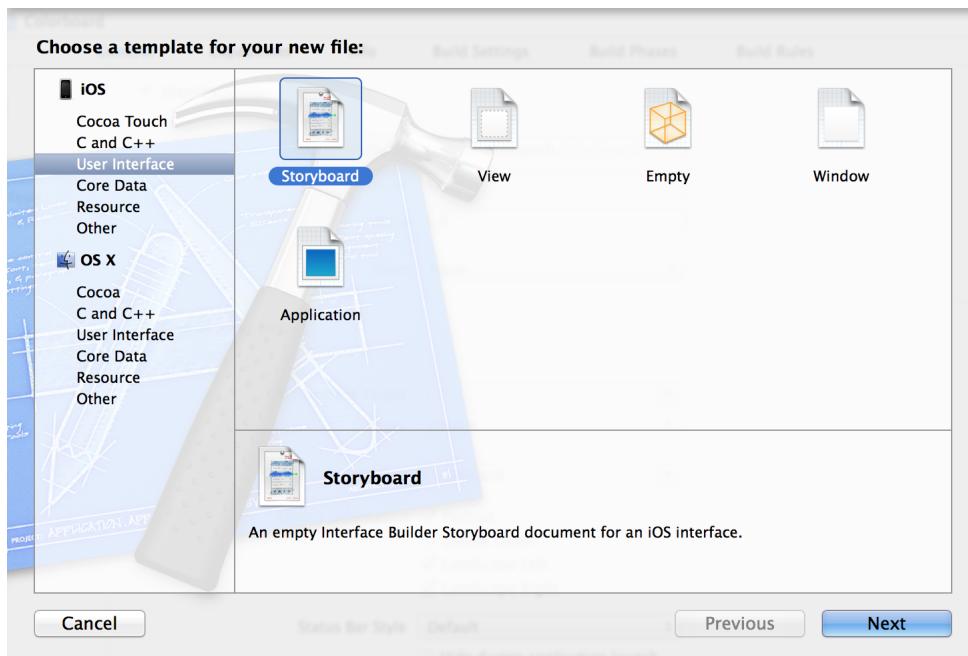
Crie um novo Empty Application (aplicativo vazio) para iOS e dê a ele o nome de Colorboard (Figure 28.1).

Figure 28.1 Criação do Colorboard



Depois, selecione New File... no menu New. Selecione User Interface na seção iOS. Depois, selecione o template Storyboard e clique em Next (Figure 28.2).

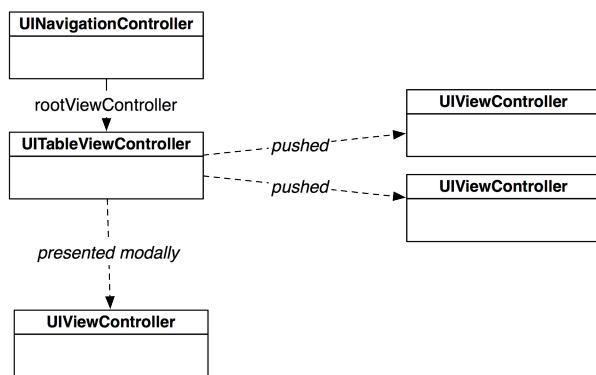
Figure 28.2 Criação de um storyboard



No próximo painel, selecione iPhone no menu pop-up Device Family e clique em Next. Depois, nomeie esse arquivo como Colorboard.

Isso criará um novo arquivo chamado `Colorboard.storyboard`, que é aberto na área do editor. Um storyboard é muito parecido com um XIB, mas ele permite que você disponha os relacionamentos entre os controladores de visão além das interfaces. O aplicativo Colorboard terá cinco controladores de visão no total, incluindo uma `UINavigationController` e uma `UITableViewViewController`. A Figure 28.3 mostra um diagrama de objetos do Colorboard.

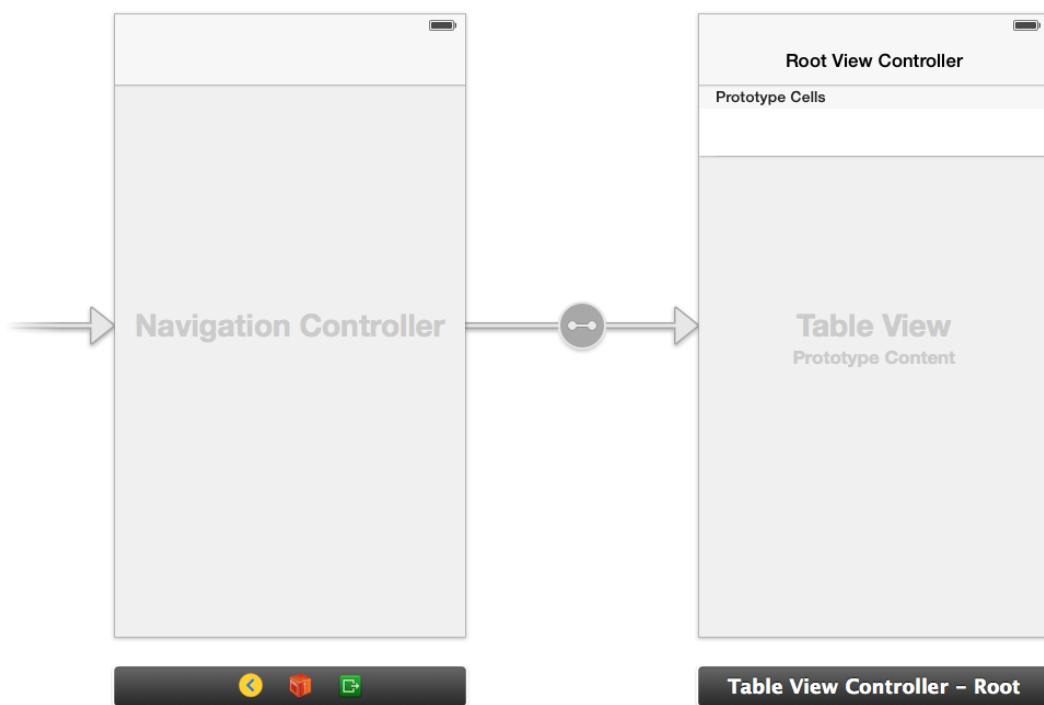
Figure 28.3 Diagrama de objetos do Colorboard



Usando um storyboard, você pode definir os relacionamentos mostrados na Figure 28.3 sem precisar escrever nenhum código.

Para começar, abra a área de utilitários e a Object Library. Arraste um Navigation Controller no canvas. O canvas agora deve ser semelhante ao da Figure 28.4.

Figure 28.4 Controlador de navegação no storyboard

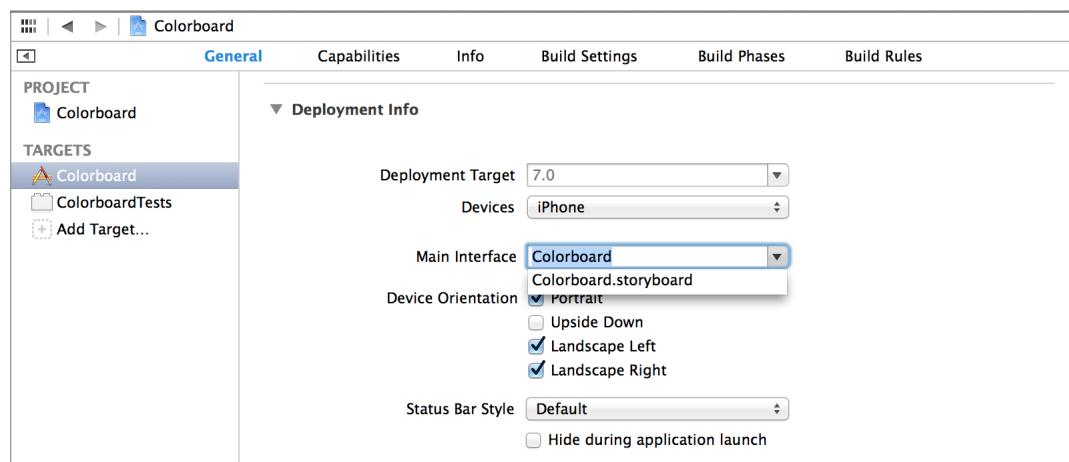


Além do objeto **`UINavigationController`** que você solicitou, o storyboard tomou a liberdade de criar três outros objetos: a visão do controlador de navegação, uma **`UITableViewController`** e a visão da **`UITableViewController`**. Além disso, a **`UITableViewController`** foi transformada no controlador de visão raiz do controlador de navegação.

As duas instâncias do controlador de visão são representadas pelas barras pretas no canvas, e suas visões são mostradas logo acima delas. Você deve configurar a visão da mesma forma que faria em um arquivo XIB normal. Para configurar você mesmo o controlador de visão, você seleciona a barra preta.

Antes de prosseguir, precisamos informar seu aplicativo sobre o arquivo de storyboard. Selecione o projeto Colorboard no navegador de projetos. Depois, selecione o destino Colorboard e a guia General. Localize o campo Main Interface e digite Colorboard (Figure 28.5) ou selecione Colorboard.storyboard no menu suspenso.

Figure 28.5 Configuração do storyboard principal



Quando um aplicativo tiver um arquivo de storyboard principal, ele carregará automaticamente esse storyboard quando for iniciado. Além de carregar o storyboard e seus controladores de visão, ele também irá criar uma

janela e definir o controlador de visão inicial do storyboard como controlador de visão raiz da janela. Você sabe qual controlador de visão é o controlador de visão inicial quando olha para o canvas do arquivo de storyboard: o controlador de visão inicial tem uma seta que se atenua na direção que aponta para ele.

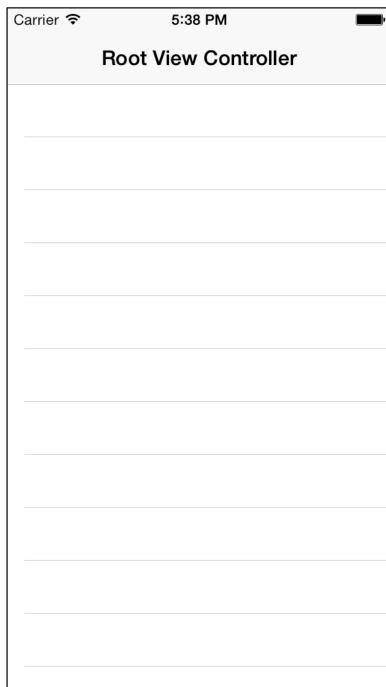
Como um arquivo de storyboard é que fornece a janela para um aplicativo, o delegate do aplicativo não precisa criar essa janela.

No `BNRAppDelegate.m`, remova o código do `application:didFinishLaunchingWithOptions:` que cria a janela.

```
- (BOOL)application:(UIApplication *)application
             didFinishLaunchingWithOptions:(NSDictionary *)launchOptions
{
    self.window = [[UIWindow alloc] initWithFrame:[[UIScreen mainScreen] bounds]];
    // Override point for customization after application launch
    self.window.backgroundColor = [UIColor whiteColor];
    [self.window makeKeyAndVisible];
    return YES;
}
```

Compile e execute o aplicativo, e você verá uma visão de um controlador de visão e uma barra de navegação que diz Root View Controller (Figure 28.6). Tudo isso veio do arquivo de storyboard – você não teve que escrever qualquer código.

Figure 28.6 Tela inicial do Colorboard



## UITableViewControllers em storyboards

Ao utilizar uma `UITableViewcontroller`, normalmente você implementa os métodos da fonte de dados apropriados para retornar o conteúdo de cada célula. Isso faz sentido quando você tem conteúdo dinâmico, tal como uma lista de itens que pode mudar, mas dá muito trabalho quando você tem uma tabela cujo conteúdo não muda nunca. Os storyboards permitem que você adicione conteúdo estático a uma visão de tabela sem ter que implementar os métodos da fonte de dados.

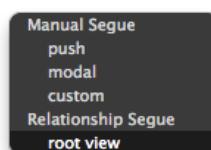
Para você ver como é fácil, vamos adicionar uma `UITableViewcontroller` ao storyboard e dar a ela um conteúdo estático. A Apple muda frequentemente os templates de arquivos, de forma que é possível que a

`rootViewController` de sua **UINavigationController** já não seja uma instância de **UITableViewController**. De qualquer forma (é bom praticar!), vamos percorrer os passos para adicionar uma.

Se já houver um segundo controlador de visão em seu storyboard próximo ao controlador de navegação, selecione a barra preta (a representação do controlador de visão em si) e exclua-a.

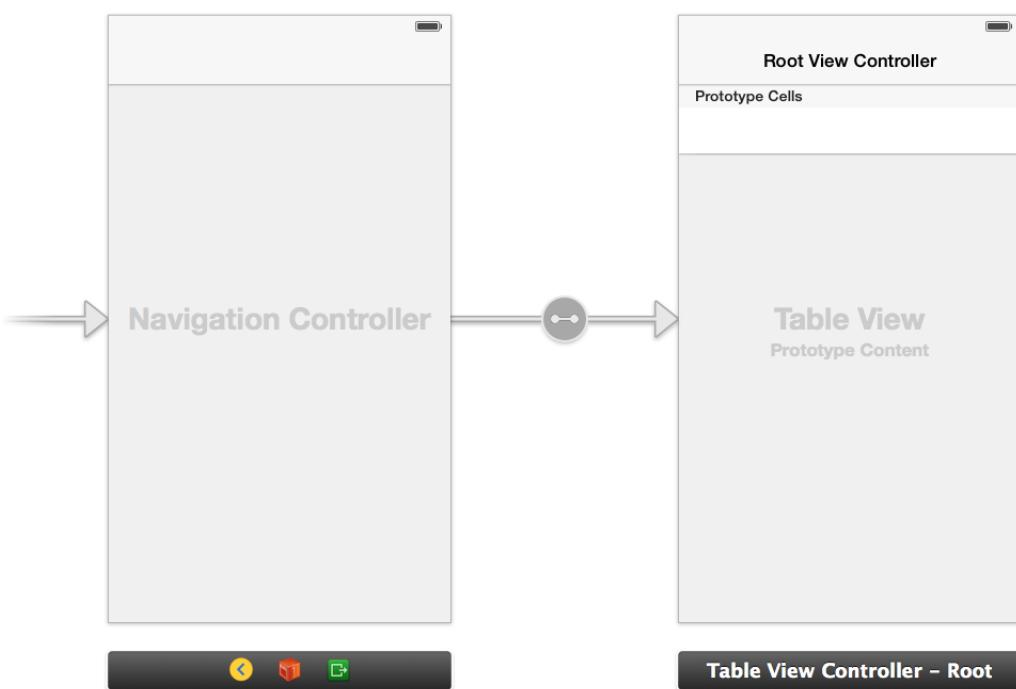
Depois, arraste uma **UITableViewController** da biblioteca para o canvas. Para definir esse controlador de visão de tabela como o controlador de visão raiz do controlador de navegação, pressione Control e arraste da visão do controlador de navegação para o controlador de visão de tabela. Solte, e no painel preto que aparece, selecione `root view` (Figure 28.7). Lembre-se de que, apesar de serem arrastadas entre visões, essas propriedades estão sendo definidas nos próprios controladores de visão.

Figure 28.7 Definição de um relacionamento



Isso estabelece a **UITableViewController** como o controlador de visão raiz da **UINavigationController**. Agora, haverá uma seta do controlador de navegação para o controlador de visão de tabela. No meio dessa seta há um ícone que representa o tipo de relacionamento entre os dois controladores de visão (Figure 28.8).

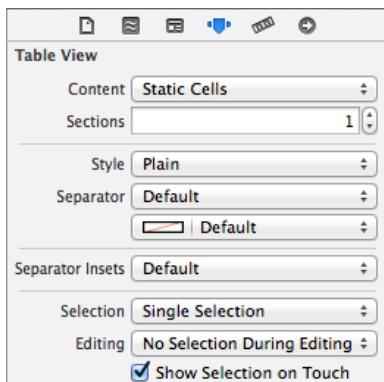
Figure 28.8 UINavigationController e UITableViewController



(Está vendo os controles de zoom no canto inferior direito do canvas? Você pode aproximar ou afastar para ver uma parte maior do canvas. Isso é especialmente útil quando se tem muitos controladores de visão. Porém, não é possível selecionar os objetos de visão quando o zoom está afastado.)

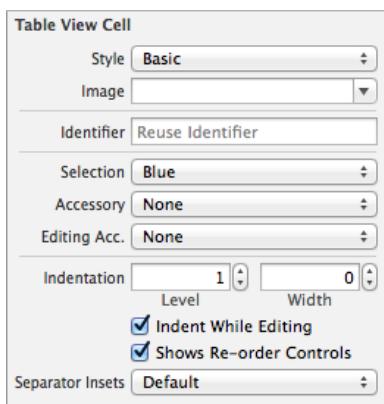
Depois, selecione o Table View da **UITableViewController**. No inspetor de atributos, altere o menu pop-up Content para **Static Cells** (Figure 28.9).

Figure 28.9 Células estáticas



Três células aparecem na visão de tabela. Agora, você pode selecionar e configurar cada uma delas individualmente. Selecione a primeira célula no topo e, no inspetor de atributos, altere seu Style para Basic (Figure 28.10).

Figure 28.10 UITableViewCellStyle básico

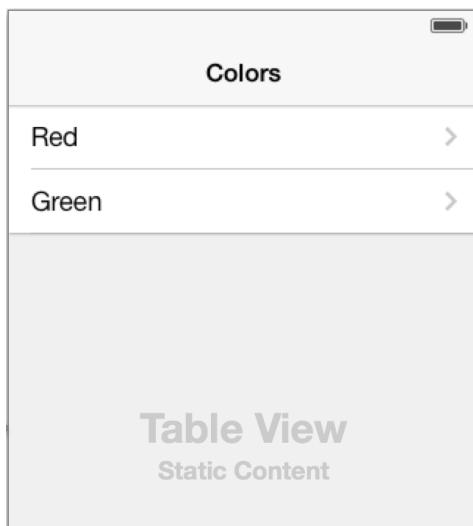


No canvas, a célula selecionada agora estará identificada como Title. Clique duas vezes no texto e altere para Red.

Reita os mesmos passos para a segunda célula, mas, desta vez, o título deve ser Green. Vamos nos livrar da terceira célula; selecione-a e pressione Delete.

Finalmente, selecione a barra de navegação – a área acima da primeira célula. Ela está presente porque o controlador de visão de tabela está embutido em um controlador de navegação. Em Attributes Inspector, mude o título para Colors. A Figure 28.11 mostra a visão de tabela atualizada.

Figure 28.11 Células configuradas



Compile e execute o aplicativo. Você verá exatamente o que determinou no arquivo de storyboard: uma visão de tabela sob uma barra de navegação. A visão de tabela é intitulada `Colors` e tem duas células nas quais se lê `Red` e `Green`. E você não teve que escrever quaisquer métodos de fonte de dados ou configurar um item de navegação.

## Segues

A maioria dos aplicativos de iOS apresentam diversos controladores de visão entre os quais os usuários navegam. Os storyboards permitem que você configure essas interações como *segues*, sem a necessidade de escrever código.

Um segue movimenta outro controlador de visão para a tela quando disparado, e é representado por uma instância de **UIStoryboardSegue**. Cada segue tem um estilo, um item de ação e um identificador. O *estilo* de um segue determina como o controlador de visão será apresentado como, por exemplo, colocado na pilha ou apresentado modalmente. O *item de ação* é o objeto de visão no arquivo de storyboard que dispara o segue, como um botão, um item de botão de barra, ou outra **UIControl**. O *identificador* é usado para acessar programaticamente o segue. É útil quando você quer ativar um segue que não vem de um item de ação, como uma agitação ou algum outro elemento de interface que não pode ser configurado no arquivo de storyboard.

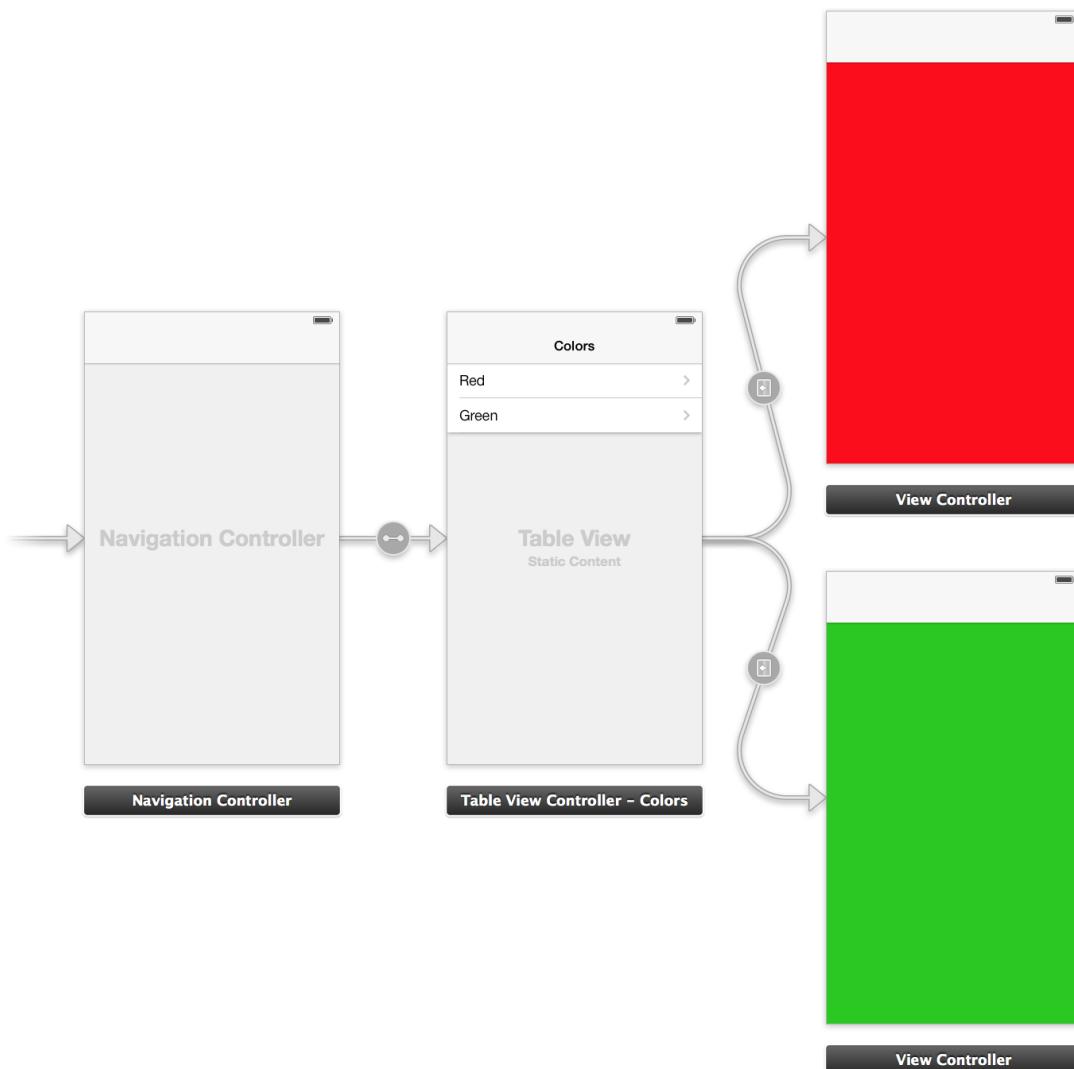
Vamos começar com dois push segues. O push segue empurra um controlador de visão para a pilha do controlador de navegação. Você precisará definir mais dois controladores de visão em nosso storyboard: um com o plano de fundo da visão em vermelho e o outro, em branco. Os segues ficam entre o controlador de visão de tabela e esses dois novos controladores de visão. Os itens de ação são as células da visão de tabela; tocar em uma célula faz com que o controlador de visão apropriado seja empurrado para a pilha do controlador de navegação.

Arraste dois objetos **UIViewController** para o canvas. Selecione a View de um dos controladores de visão e, no inspetor de atributos, altere a cor do plano de fundo para vermelho. Faça o mesmo com a visão do outro controlador de visão para configurar a cor do plano de fundo para vermelho.

Depois, selecione a célula `Red`. Pressione Control e arraste para o controlador de visão cuja visão tem o plano de fundo vermelho. Um painel preto intitulado **Storyboard Segues** aparece. Esse painel lista os estilos possíveis para esse segue. Selecione **Push**.

Depois, selecione a célula `Green`, pressione Control e arraste para o outro controlador de visão. Seu canvas deve ficar como na Figure 28.12.

Figure 28.12 Configuração de dois segues



Observe as setas que saem do controlador de visão de tabela e vão para os outros dois controladores de visão. Cada uma delas é um segue. O ícone no círculo informa que esses segues são push segues.

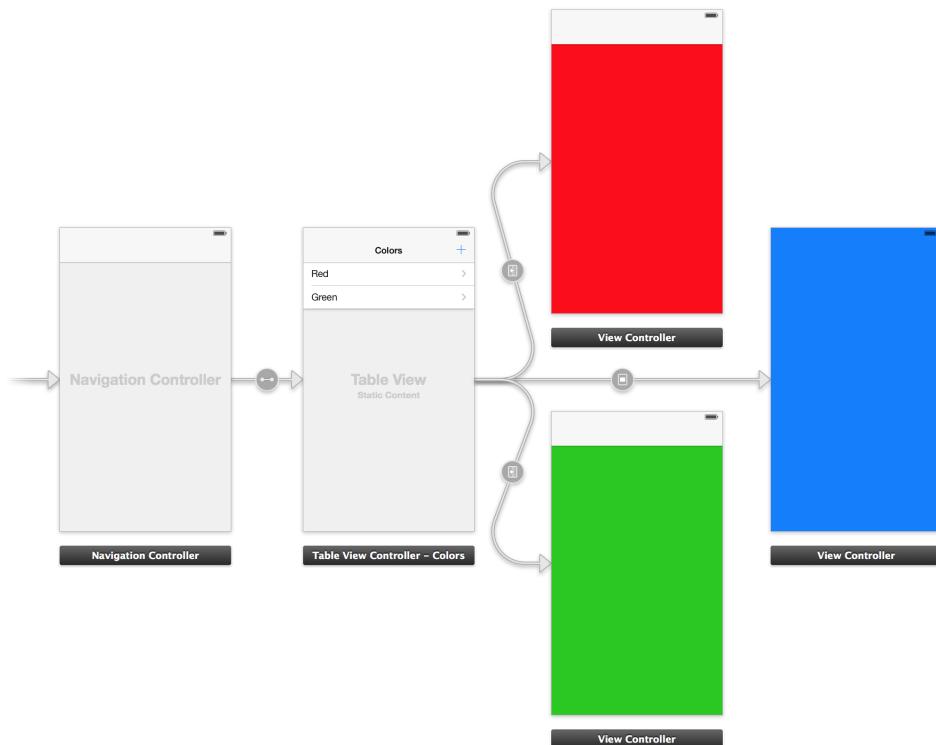
Compile e execute o aplicativo. Toque em cada linha e você será levado ao controlador de visão apropriado. Você pode até voltar na pilha de navegação para o controlador de visão de tabela, como esperado. E o melhor de tudo isso? Você ainda não escreveu nenhum código.

Observe que os push segues só funcionam se a origem do segue estiver dentro de um controlador de navegação. Felizmente, a origem desses segues é o controlador de visão de tabela, que atende a esse requisito.

Agora vamos examinar outro estilo de segue: um segue Modal. Arraste uma nova **UIViewController** para o canvas. Mude a cor de fundo da sua visão para azul. Você quer que o item de ação desse segue seja um item de botão de barra no item de navegação do controlador de visão de tabela.

Arraste um **Bar Button Item** da biblioteca para o canto direito da barra de navegação no topo da visão do controlador de visão de tabela. No inspetor de atributos, altere o seu **Identifier** para **Add**. Depois, pressione **Control** e arraste desse item de botão de barra para o controlador de visão que você acaba de soltar no canvas. Selecione **Modal** no painel preto. O canvas do storyboard agora é semelhante ao da Figure 28.13. (Observe que o ícone do segue modal é diferente do ícone dos push segues.)

Figure 28.13 Um segue modal

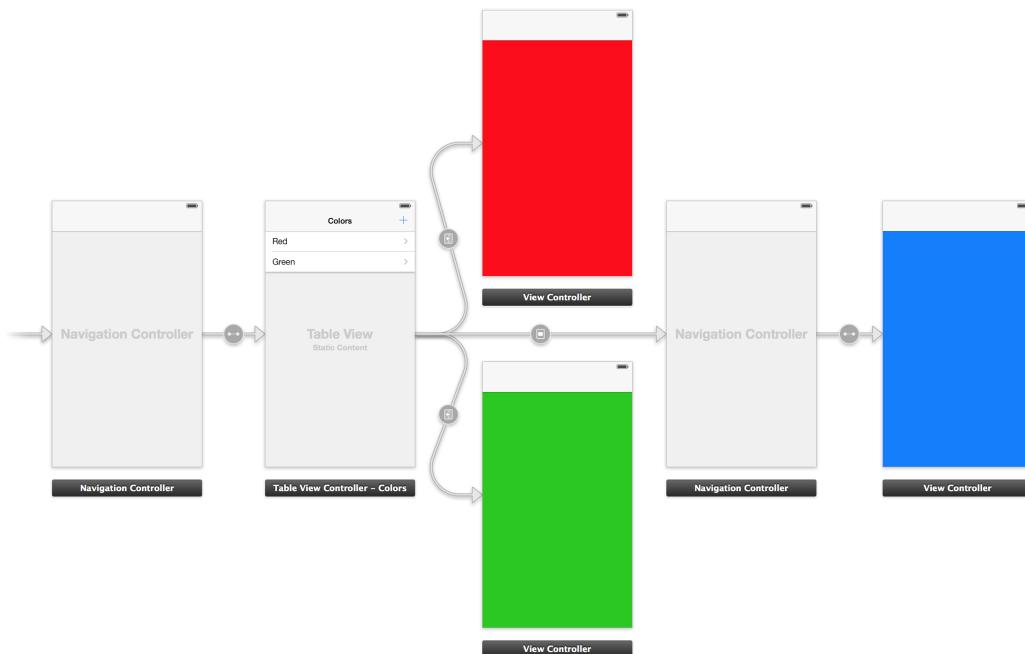


Compile e execute o aplicativo. Toque no item de botão de barra, e um controlador de visão com uma visão azul aparecerá na tela. Tudo está indo bem – exceto o fato de que você não pode dispensar esse controlador de visão.

Você irá dispensar o controlador de visão a partir de uma **UIBarButtonItem** na barra de navegação onde está escrito **Done**. No momento, o controlador de visão modal está sendo apresentado por ele mesmo, de forma que não há barra de navegação para o item de botão de barra. Para corrigir isso, arraste uma **UINavigationController** para o canvas e exclua a **UITableViewController** (ou qualquer que seja o segundo controlador de visão que a Apple forneceu o controlador de navegação).

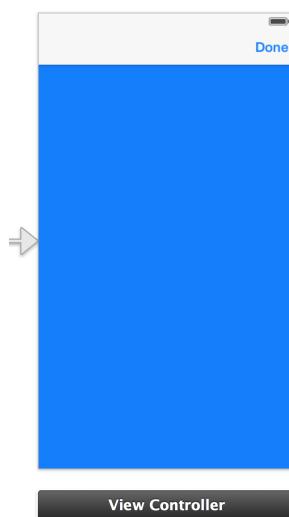
Exclua o segue modal existente e, no lugar dele, faça com que o item + ative um segue modal para o controlador de navegação. O controlador de visão azul existente deve ser a raiz do controlador de navegação. Seu storyboard deve ficar semelhante ao da Figure 28.14.

Figure 28.14 Adição no controlador de navegação



Agora que o controlador de visão modal está dentro de um controlador de navegação, ele tem uma barra de navegação em seu topo. Arraste um item de botão de barra para o lado direito dessa barra de navegação. No inspetor de atributos, altere seu Identifier para Done. O controlador de visão deve ser semelhante ao da Figure 28.15.

Figure 28.15 Botão Done



Este ponto é o mais longe a que você pode chegar sem escrever qualquer código. Você precisará escrever um método para dispensar o controlador de visão modal e então conectar esse método ao botão Done.

Neste momento, todos os controladores de visão do storyboard são uma instância padrão da **UIViewController** ou uma de suas subclasses padrão. Não podemos escrever código para nenhum deles do jeito que estão. Para escrever código para um controlador de visão em um storyboard, você precisa criar uma subclasse de **UIViewController** e especificar no storyboard que o controlador de visão é uma instância da sua subclasse.

Vamos criar uma nova subclasse de **UIViewController** para ver como isso funciona. Crie uma nova subclasse **NSObject** e nomeie-a como **BNRColorViewController**.

No `BNRColorViewController.h`, altere a superclasse para **UIViewController**.

```
@interface BNRColorViewController : NSObject
@interface BNRColorViewController : UIViewController

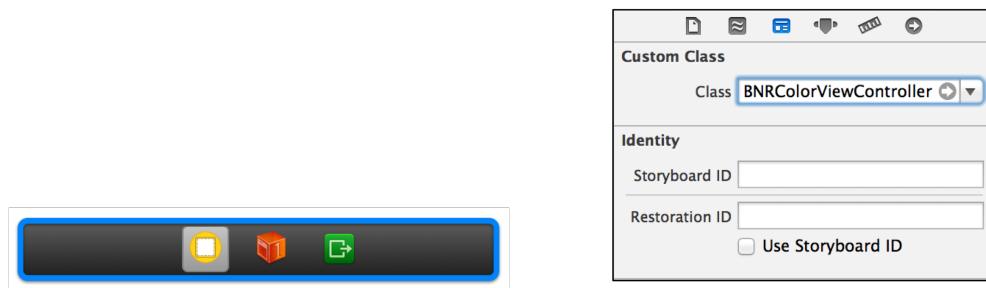
@end
```

Depois, no `BNRColorViewController.m`, implemente um método para dispensar a si próprio.

```
- (IBAction)dismiss:(id)sender
{
    [self.presentingViewController dismissViewControllerAnimated:YES
                                                    completion:nil];
}
```

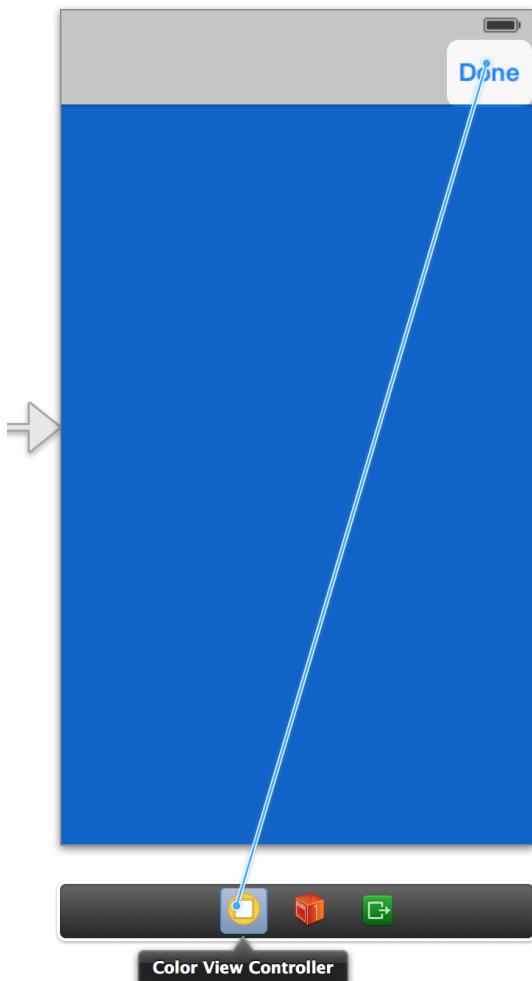
Abra o `Colorboard.storyboard` novamente. Selecione a barra preta sob o controlador de visão apresentado modalmente (azul). No inspetor de identidade, altere Class para **BNRColorViewController** (Figure 28.16).

Figure 28.16 Alteração do controlador de visão para **BNRColorViewController**



Agora, depois de se certificar que você está com o zoom aproximado, selecione o botão Done. Pressione Control e arraste do botão para esse ícone do controlador de visão, e depois solte; quando o painel aparecer, selecione o método **dismiss:** (Figure 28.17).

Figure 28.17 Definição de outlets e ações em um storyboard

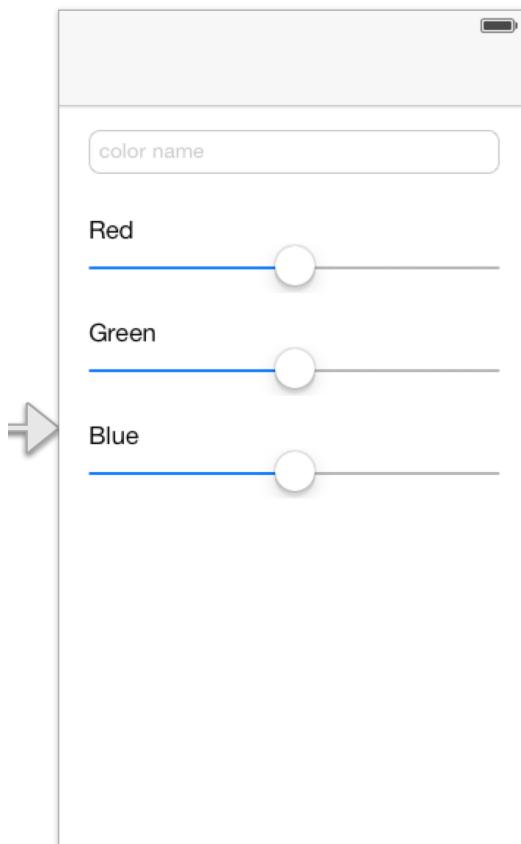


Esse botão agora está programado para enviar a mensagem `dismiss`: a sua `BNRColorViewController` sempre que for pressionado. Compile e execute o aplicativo, apresente a `BNRColorViewController` e, em seguida, toque no botão Done. Voilà!

## Permitindo mudanças de cor

Você agora estenderá o aplicativo Colorboard para permitir que o usuário escolha uma cor e salve-a em uma lista de cores favoritas.

De volta ao `Colorboard.storyboard`, adicione uma `UITextField`, três objetos `UILabel` e três objetos `UISlider` à visão de `BNRColorViewController`, de forma que fique semelhante à Figure 28.18.

Figure 28.18 Configuração da visão de **BNRColorViewController**

Vamos fazer com que a cor de fundo da visão de **BNRColorViewController** corresponda aos valores do controle deslizante. No **BNRColorViewController.m**, adicione outlets ao campo de texto e três rótulos na extensão de classe.

```
@interface BNRColorViewController : UIViewController

@property (nonatomic, weak) IBOutlet UITextField *textField;

@property (nonatomic, weak) IBOutlet UISlider *redSlider;
@property (nonatomic, weak) IBOutlet UISlider *greenSlider;
@property (nonatomic, weak) IBOutlet UISlider *blueSlider;

@end

@implementation
```

Todos os três controles deslizantes ativarão o mesmo método quando seus valores forem alterados. Implemente esse método no **BNRColorViewController.m**.

```
- (IBAction)changeColor:(id)sender
{
    float red = self.redSlider.value;
    float green = self.greenSlider.value;
    float blue = self.blueSlider.value;
    UIColor *newColor = [UIColor colorWithRed:red
                                         green:green
                                         blue:blue
                                         alpha:1.0];
    self.view.backgroundColor = newColor;
}
```

Agora abra o `Colorboard.storyboard` e conecte os outlets do Color View Controller ao campo de texto e aos três rótulos. Depois, pressione Control e arraste de cada controle deslizante até o Color View Controller e conecte cada um deles ao método `changeColor:`.

Compile e execute o aplicativo. Mover os controles deslizantes fará com que a visão fique com a cor de fundo correspondente.

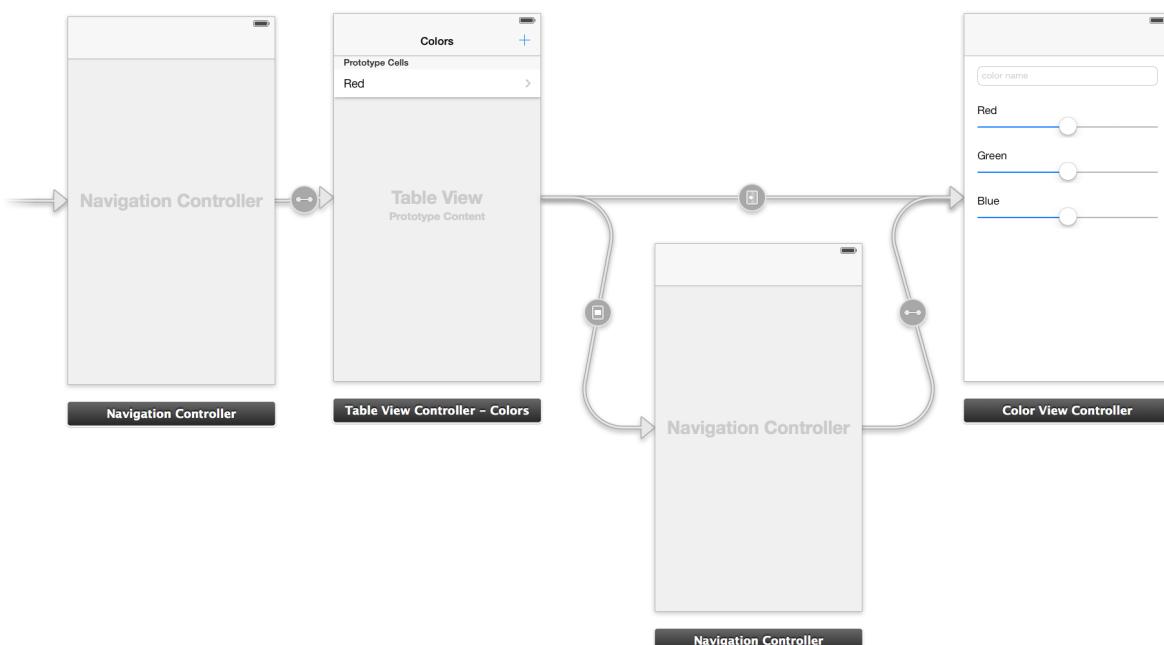
## Repassando dados

Como discutimos no Chapter 10, muitas vezes é necessário que os controladores de visão repassem os dados. Para mostrar isso, você vai fazer com que o Colorboard tenha uma lista de cores favoritas que possa ser editada pesquisando o `BNRColorViewController` que você acabou de configurar.

Em vez de usar células estáticas para a `UITableView`, você voltará a usar *protótipos dinâmicos*. Por causa disso, você terá de implementar os métodos de fonte de dados para a visão de tabela. Células de protótipo permitem que você configure as várias células que você vai querer retornar nos métodos de fonte de dados e atribua um identificador de reutilização a cada uma.

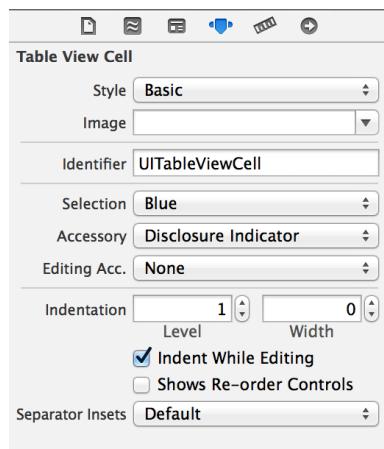
No `Colorboard.storyboard`, exclua os controladores de visão Red e Green que estão sendo empurrados da `UITableView`. Depois, selecione a visão de tabela e abra o inspetor de atributos. Mude o tipo Content para Dynamic Prototypes e exclua a segunda `UITableViewCell`. O storyboard deve ser semelhante ao da Figure 28.19.

Figure 28.19 Storyboard de protótipos dinâmicos



Depois, selecione a `UITableViewCell` e configure o identificador de reutilização como `UITableViewCell` (Figure 28.20).

Figure 28.20 Identificador de reutilização de **UITableViewCell**



Para fornecer a esse controlador de visão de tabela dados para a sua visão de tabela, você precisará criar uma nova subclasse **UITableViewController**. Crie uma nova subclasse **NSObject** chamada **BNRPaletteViewController**.

No **BNRPaletteViewController.h**, altere a superclasse para **UITableViewController**.

```
@interface BNRPaletteViewController : NSObject
@interface BNRPaletteViewController : UITableViewController
@end
```

No **BNRPaletteViewController.m**, adicione uma **NSMutableArray** para a extensão de classe.

```
#import "BNRPaletteViewController.h"

@interface BNRPaletteViewController ()  
@property (nonatomic) NSMutableArray *colors;  
@end  
@implementation BNRPaletteViewController
```

A seguir, implemente os métodos de fonte de dados da visão de tabela no **BNRPaletteViewController.m**.

```
- (NSInteger)tableView:(UITableView *)tableView  
    numberOfRowsInSection:(NSInteger)section  
{  
    return [self.colors count];  
}  
  
- (UITableViewCell *)tableView:(UITableView *)tableView  
    cellForRowAtIndexPath:(NSIndexPath *)indexPath  
{  
    UITableViewCell *cell =  
        [tableView dequeueReusableCellWithIdentifier:@"UITableViewCell"  
            forIndexPath:indexPath];  
  
    return cell;  
}
```

Depois, crie uma nova subclasse **NSObject** chamada **BNRColorDescription**, que representará uma cor definida pelo usuário.

No **BNRColorDescription.h**, adicione propriedades para uma **UIColor** e um nome.

```
@interface BNRColorDescription : NSObject
@property (nonatomic) UIColor *color;
@property (nonatomic, copy) NSString *name;
@end
```

Depois, no `BNRColorDescription.m`, sobrescreva o `init` para definir valores padrão para essas propriedades.

```
@implementation BNRColorDescription
- (instancetype)init
{
    self = [super init];
    if (self) {
        _color = [UIColor colorWithRed:0
                                green:0
                                 blue:1
                                alpha:1];
        _name = @"Blue";
    }
    return self;
}
@end
```

Para testar se o código funciona, vamos adicionar uma nova `BNRColorDescription` ao array `colors` de `BNRPaletteViewController`.

No topo do `BNRPaletteViewController.m`, importe o `BNRColorDescription.h`. Depois, sobrescreva o acessor `colors` para instanciar vagarosamente o array e adicione uma nova `BNRColorDescription` ao array.

```
#import "BNRPaletteViewController.h"
#import "BNRColorDescription.h"

@implementation BNRPaletteViewController
- (NSMutableArray *)colors
{
    if (!_colors) {
        _colors = [NSMutableArray array];
        BNRColorDescription *cd = [[BNRColorDescription alloc] init];
        [_colors addObject:cd];
    }
    return _colors;
}
```

Além disso, atualize o método de fonte de dados no `BNRPaletteViewController.m` para mostrar o nome da cor.

```
- (UITableViewCell *)tableView:(UITableView *)tableView
    cellForRowAtIndexPath:(NSIndexPath *)indexPath
{
    UITableViewCell *cell =
        [tableView dequeueReusableCellWithIdentifier:@"UITableViewCell"
                                    forIndexPath:indexPath];
    BNRColorDescription *color = self.colors[indexPath.row];
    cell.textLabel.text = color.name;
    return cell;
}
```

Compile e execute o aplicativo. Você deve ser capaz de fazer a busca, mas há dois problemas. Primeiro, a cor “Blue” não é passada para a `BNRColorDescription`. Segundo, o controlador de visão atualmente está mostrando o botão Done em conjunto com o botão Back. Idealmente, o botão Done só estaria presente se você estivesse criando uma nova cor e apresentando esse controlador de visão modalmente. Para corrigir ambos os problemas, você terá de ser capaz de passar dados entre controladores de visão quando ocorrerem segues.

Antes de prosseguirmos, abra o `BNRColorViewController.h` e adicione duas novas propriedades: uma que determina se você está editando uma cor nova ou existente, e outra que indica que cor você está editando. Não esqueça de importar o `BNRColorDescription.h` e o `BNRViewController.h` no topo.

```
#import "BNRColorDescription.h"
#import "BNRViewController.h"

@interface BNRColorViewController : UIViewController

@property (nonatomic) BOOL existingColor;
@property (nonatomic) BNRColorDescription *colorDescription;

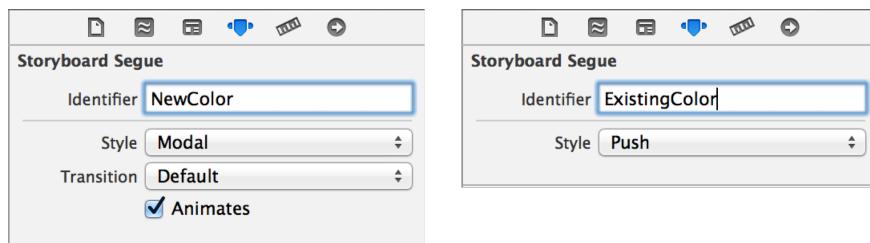
@end
```

Sempre que um segue é ativado em um controlador de visão, ele recebe a mensagem `prepareForSegue:sender:`. Esse método dá a você a `UIStoryboardSegue`, que fornece informações sobre qual segue está acontecendo, e o `sender`, que é o objeto que ativou o segue (uma `UIBarButtonItem` ou uma `UITableViewCell`, por exemplo).

O segue fornece a você três informações para usar: o controlador de visão fonte (que origina o segue), o controlador de visão de destino (para o qual você está fazendo o segue) e o identificador do segue. O identificador é o que possibilita que você diferencie os vários segues. Vamos dar a seus dois segues identificadores úteis.

Abra o `Colorboard.storyboard` novamente. Selecione o segue modal e abra seu inspetor de atributos. Para o identificador, digite `NewColor`. A seguir, selecione o push segue e dê a ele o identificador `ExistingColor`. O inspetor de atributos de ambos os segues é mostrado na Figure 28.21.

Figure 28.21 Identificadores de segue



Com os seus segues identificados, agora você pode passar os seus objetos de cor. Abra o `BNRPaletteViewController.m` e implemente `prepareForSegue:sender:`:

```
- (void)prepareForSegue:(UIStoryboardSegue *)segue sender:(id)sender
{
    if ([segue.identifier isEqualToString:@"NewColor"]) {

        // If we are adding a new color, create an instance
        // and add it to the colors array
        BNRColorDescription *color = [[BNRColorDescription alloc] init];
        [self.colors addObject:color];

        // Then use the segue to set the color on the view controller
        UINavigationController *nc =
            (UINavigationController *)segue.destinationViewController;
        BNRColorViewController *mvc =
            (BNRColorViewController *)[nc topViewController];
        mvc.colorDescription = color;
    }
    else if ([segue.identifier isEqualToString:@"ExistingColor"]) {

        // For the push segue, the sender is the NSIndexPath
        NSIndexPath *ip = [self.tableView indexPathForCell:sender];
        BNRColorDescription *color = self.colors[ip.row];

        // Set the color, and also tell the view controller that this
        // is an existing color
        BNRColorViewController *cvc =
            (BNRColorViewController *)segue.destinationViewController;
        cvc.colorDescription = color;
        cvc.existingColor = YES;
    }
}
```

Primeiro, o identificador do segue é verificado para determinar qual segue está ocorrendo. Se o botão + foi pressionado, o segue “NewColor” é ativado, de forma que você cria uma nova **BNRColorDescription** e dá ela para a **BNRColorViewController**.

Se você tocar em uma cor existente, o segue “ExistingColor” é ativado. Observe que, quando uma **UITableViewCell** ativa um segue, este é enviado como o argumento **sender**, e você pode usar isso para determinar qual caminho índice foi selecionado. A cor que foi pressionada é, então, passada para **BNRColorViewController**.

(Por que a **destinationViewController** da “NewColor” é uma **UINavigationController** quando ela é uma **BNRColorViewController** de “ExistingColor”? Volte a examinar o arquivo de storyboard e você perceberá que o segue modal apresenta uma nova **UINavigationController**, enquanto o push segue está empurrando um controlador de visão sobre uma pilha de controladores de navegação existente.)

Você precisa amarrar algumas pontas soltas da **BNRColorViewController**: o botão Done não deverá estar lá se você estiver visualizando uma cor existente, a cor de fundo e os controles deslizantes devem estar configurados corretamente, e você precisa salvar os novos valores que o usuário escolheu quando a **BNRColorViewController** desaparecer (seja dispensando o controlador de visão modal ou diretamente da pilha de controladores de navegação).

Abra o **BNRColorViewController.m** e sobrescreva **viewWillAppear**: para se livrar do botão Done se **existingColor** for YES.

```
- (void)viewWillAppear:(BOOL)animated
{
    [super viewWillAppear:animated];

    // Remove the 'Done' button if this is an existing color
    if (self.existingColor) {
        self.navigationItem.rightBarButtonItem = nil;
    }
}
```

Depois, ainda no **BNRColorViewController.m**, sobrescreva **viewDidLoad** para definir a cor de fundo inicial, os valores do controle deslizante e nome da cor.

```
- (void)viewDidLoad
{
    [super viewDidLoad];

    UIColor *color = self.colorDescription.color;

    // Get the RGB values out of the UIColor object
    float red, green, blue;
    [color getRed:&red
              green:&green
                blue:&blue
               alpha:nil];

    // Set the initial slider values
    self.redSlider.value = red;
    self.greenSlider.value = green;
    self.blueSlider.value = blue;

    // Set the background color and text field value
    self.view.backgroundColor = color;
    self.textField.text = self.colorDescription.name;
}
```

Finalmente, salve os valores quando a visão estiver desaparecendo.

```
- (void)viewWillDisappear:(BOOL)animated
{
    [super viewWillDisappear:animated];

    self.colorDescription.name = self.textField.text;
    self.colorDescription.color = self.view.backgroundColor;
}
```

Compile e execute o aplicativo e as cores deverão ser mostradas e salvas corretamente.

## Mais sobre storyboards

Neste exercício, você criou um storyboard, configurou alguns controladores de visão, distribuiu as respectivas interfaces e criou alguns segues entre eles. Essa é a ideia básica por trás dos storyboards, e embora existam mais alguns tipos de segues e de controladores de visão que possam ser configurados, você já tem uma boa noção. Um storyboard substitui linhas de código.

Por exemplo, um push segue substitui este código:

```
- (void)tableView:(UITableView *)tableView
didSelectRowAtIndexPath:(NSIndexPath *)ip
{
    UIViewController *vc = [[UIViewController alloc] init];
    [self.navigationController pushViewController:vc];
}
```

Embora pareça legal, na nossa opinião, o uso de storyboards não ajuda muito. Vamos falar dos prós e dos contras. Primeiro, os prós:

- Os storyboards podem ser usados para demonstrar facilmente o fluxo de um aplicativo para um cliente ou colega.
- Os storyboards removem um pouco do código mais simples de seus arquivos-fonte.
- As tabelas com conteúdo estático ficam fáceis de criar.
- As células de protótipo substituem a necessidade de criar XIBs separados para células de visão de tabela personalizada.
- Os storyboards com certeza são bonitos.

Mas os contras, infelizmente, superam os prós:

- Os storyboards não são propícios para trabalhar em equipe. Geralmente, uma equipe de programadores de iOS divide o trabalho, fazendo com que cada membro se concentre em um controlador de visão específico. Com o storyboard, todo mundo precisa trabalhar no mesmo arquivo de storyboard. Isso pode levar rapidamente a um congestionamento e a dificuldades com o controle de versões.
- Os storyboards perturbam o fluxo de programação. Digamos que você esteja escrevendo um controlador de visão e adicionando código para um botão que apresenta um controlador de visão modalmente. Você pode fazer isso facilmente em código: **alloc** e **init** o controlador de visão, e envie **presentViewController:animated:completion:** para **self**. Com storyboards, você precisa carregar o arquivo de storyboard, arrastar algumas coisas para o canvas, definir a Class no inspetor de identidade, conectar o segue e depois configurar o segue.
- Os storyboards prejudicam a flexibilidade e o controle em prol da facilidade de uso. O trabalho necessário para adicionar funcionalidade avançada à funcionalidade básica de um storyboard é normalmente superior ao trabalho necessário para obter funcionalidade básica e avançada em código.
- Os storyboards sempre criam novas instâncias de controladores de visão. Cada vez que você efetua um segue, uma nova instância do controlador de visão de destino é criada. Às vezes, porém, você quer manter um controlador de visão, em vez de destruí-lo a cada vez que ele desaparece da tela. O uso de storyboard não permite isso.

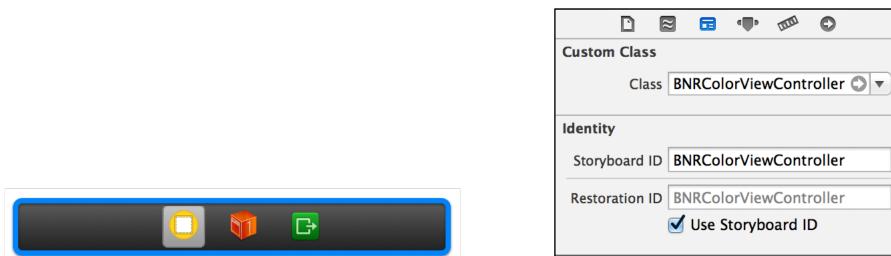
Em termos gerais, os storyboards tornam o código fácil mais fácil e o código difícil mais difícil. Não os utilizamos neste livro, e geralmente não os utilizamos para escrever nossos próprios aplicativos. Contudo, a Apple parece estar incentivando-os cada vez mais em cada versão do Xcode, de forma que um dia você talvez possa decidir que um aplicativo em particular seria beneficiado com storyboards.

## Para os mais curiosos: restauração de estado

Discutimos como trabalhar com o sistema de restauração de estado no Chapter 24, mas o aplicativo Homepwner não utilizou storyboards. Vamos observar como a restauração de estado funciona ao utilizar storyboards.

Os storyboards assumem para você grande parte do código de restauração do estado padrão. Dentro de um arquivo de storyboard, o identificador de restauração de cada controlador de visão pode ser configurado. Normalmente, o identificador de restauração é configurado como igual ao identificador do storyboard (que você não precisava usar neste capítulo). Para que a restauração de estado funcione corretamente, cada controlador de visão deverá ter tanto um identificador de storyboard quanto um de restauração (Figure 28.22).

Figure 28.22 Identificador de restauração



Já que a restauração de estado é opcional, você ainda terá de sobrescrever os dois métodos de delegate de aplicativo para informar o sistema que você quer que os estados sejam salvos e restaurados.

```
- (BOOL)application:(UIApplication *)application
shouldSaveApplicationState:(NSCoder *)coder
{
    return YES;
}

- (BOOL)application:(UIApplication *)application
shouldRestoreApplicationState:(NSCoder *)coder
{
    return YES;
}
```

Finalmente, cada uma de suas subclasses de **UIViewController** precisará implementar o método de protocolo **UIViewControllerAnimatedRestoration** para retornar uma instância do controlador de visão apropriado. Já que você está trabalhando com storyboards, você deixará o storyboard instanciar o controlador de visão para você. Veja um exemplo:

```
+ (UIViewController *)viewControllerWithRestorationIdentifierPath:(NSArray *)path
                                                       coder:(NSCoder *)coder
{
    BNRCOLORViewController *vc = nil;

    UIStoryboard *storyboard =
        [coder decodeObjectForKey:UIStateRestorationViewControllerStoryboardKey];
    if (storyboard)
    {
        vc = (BNRCOLORViewController *)[storyboard
            instantiateViewControllerWithIdentifier:@"BNRCOLORViewController"];
        vc.restorationIdentifier = [identifierComponents lastObject];
        vc.restorationClass = [BNRCOLORViewController class];
    }
    return vc;
}
```

A **NSCoder** codifica automaticamente uma referência ao storyboard que você pode extrair utilizando a chave **UIStateRestorationViewControllerStoryboardKey**. Você pode, então, utilizar o storyboard para instanciar o controlador de visão apropriado, passando o identificador de storyboard correto.

Fora isso, o restante da implementação da restauração de estado é igual. Se os controladores de visão precisarem salvar qualquer informação, eles precisarão implementar os métodos **encodeRestorableStateWithCoder:** e **decodeRestorableStateWithCoder:**.

Esteja você utilizando ou não storyboards, a restauração de estado é fácil de implementar e propicia uma melhor experiência para seus usuários. Seja um desenvolvedor elegante: implemente já a restauração de estado em seus aplicativos!

# 29

## Posfácio

Bem-vindo ao final do livro! Você deve estar orgulhoso de todo o seu trabalho e de tudo o que aprendeu. Agora, temos uma boa e uma má notícia:

- *A boa notícia:* tudo o que deixa os programadores confusos no que se refere à plataforma do iOS ficou para trás para você agora. Você é um desenvolvedor para iOS.
- *A má notícia:* provavelmente você não seja um desenvolvedor para iOS muito bom.

### Quais são os próximos passos

Chegou a hora de cometer alguns erros, ler algumas documentações realmente cansativas e ser humilhado pelos especialistas impiedosos que ridicularizarão suas dúvidas. Veja aqui nossas recomendações:

*Escreva aplicativos agora.* Se você não usar imediatamente o que aprendeu, o conhecimento perderá a intensidade. Exercite e amplie seu conhecimento. Agora.

*Aprofunde-se.* Este livro deu mais atenção ao volume do que ao teor em si; qualquer capítulo teria se expandido e virado um livro. Encontre um tópico que ache interessante e realmente se aprofunde nele – faça alguns experimentos, leia documentos da Apple sobre o tópico, leia uma publicação em um blog ou no StackOverflow.

*Conecte-se.* Há encontros de desenvolvedores de iOS na maioria das cidades, e as palestras são sempre muito boas. Há grupos de discussão on-line. Se você estiver fazendo um projeto, encontre pessoas que possam ajudá-lo: designers, testers (também conhecidos como cobaias) e outros desenvolvedores.

*Cometa erros e corrija-os.* Você aprenderá muito com os dias em que disser: “Este aplicativo mais parece um monte de lixo! Vou apagar tudo e escrevê-lo novamente, com uma arquitetura que faça sentido.” Os programadores mais educados chamam isso de *refatoração*.

*Retribua.* Compartilhe o conhecimento. Responda a uma pergunta estúpida com educação. Forneça códigos.

### A minha propaganda

Você pode nos encontrar no Twitter. Lá, manteremos você informado sobre programação e o dia a dia: @aaronthillegass, @cbkeur e @joeconwaystk.

Fique atento aos futuros guias da Big Nerd Ranch. Oferecemos também cursos de uma semana para desenvolvedores. E, se você precisa apenas de algum código escrito, fazemos contratos de programação. Para obter mais informações, acesse nosso site em: <http://www.bignerdranch.com/>.

É você, querido leitor, que possibilita nossas vidas de escrever, codificar e ensinar. Então, muito obrigado por adquirir nosso livro.



# Index

## Symbols

#import, 37, 55  
#pragma mark, 203, 203  
.h arquivos, 35  
.xcassets (catálogo de ativos), 21  
.xcdatamodeld (arquivo do modelo de dados), 372  
@prefixo  
    criar dicionários com, 196  
@autoreleasepool, 74  
@class, 143  
@end, 35  
@implementation, 36  
@import, 55  
@interface, 99  
    em arquivos de cabeçalho, 35, 99  
@optional, 130  
@property (propriedade), 66  
@protocol, 130

## A

accessor methods,  
    (see also properties)  
**addKeyframeWithRelativeStartTime:**  
**relativeDuration:animations:**, 424, 424  
**addObject:**, 30  
**addSubview:**, 79, 81  
ajustes de compilação, 244-247, 245  
**alloc**, 26-26  
analizador estático, 242-243, 245  
analisar (código), 242-243, 245  
análise de geração, 238, 238  
animações  
    básicas, 421-423  
    com blocos, 422-426  
    funções de tempo, 422-423  
    mola, 425-426  
    quadro-chave, 423-425  
animações básicas, 421-423  
animações de mola, 425-426  
animações de quadro-chave, 423-425  
**animateKeyframesWithDuration:delay:options:**  
**animations:completion:**, 424  
**animateWithDuration:animations:**, 422-423  
**animateWithDuration:delay:options:**  
**animations:completion:**, 422  
antiserrilhado (anti-aliasing), 123  
aparência do teclado, 127  
aplicações universais  
    definição da família de dispositivos, 369  
aplicativo Homepwner

adição de restrições do Auto Layout, 272-279  
adicionar apresentação modal, 287-293  
adicionar armazenamento de itens, 142-145  
adicionar Dynamic Type, 337-346  
adicionar imagens de itens, 185-201  
adicionar interface drill-down, 167-183, 169  
adicionar preferências ao, 416-420  
adicionar restauração de estado, 391-401  
adicionar restrições com Auto Layout, 251-269  
adicionar um armazenamento de imagens, 193  
armazenar imagens, 309-311  
customizar células, 319-332  
diagramas de objetos, 142, 169  
habilitar edição, 155-164  
localizar, 403-411  
mudando para o Core Data, 371  
reutilizar a classe **BNRItem**, 141  
universalizar, 249-250  
aplicativo HypnoNerd  
    adicionar controlador de barra de guias, 114-118  
    adicionar segundo controlador de visão, 109-114  
    adicionar uma notificação local, 119-119  
    criar, 105-108  
aplicativo Hypnosister  
    criar **BNRHypnosisView**, 77-78  
    diagrama de objeto, 100  
    lidar com evento de toque, 97-98  
    rolagem, 100-103  
aplicativo Nerdfeed  
    adicionar **UIWebView**, 357-359  
    obter dados, 352-353  
    usar **UISplitViewController**, 363-365  
aplicativo Quiz, 1-23  
aplicativo RandomItems  
    criar, 28-33  
    criar classe **BNRItem**, 34-49  
aplicativo Settings, 304  
aplicativo TouchTracker  
    reconhecer gestos, 219-228  
    traçar linhas, 208-215  
aplicativos  
    ajustes de compilação para, 244-247  
    armazenamento de dados, 389  
    construir, 19, 408  
    criar perfis, 233-234  
    armazenamento de dados, 303-304  
    destinos e, 244  
    diretórios em, 303-305  
    executar no iPad, 2  
    executar no simulador, 19  
    ícones para, 21-22  
    imagens de abertura para, 22  
    implementar, 20-21  
    limpar, 408  
    múltiplos threads em, 357  
    otimizar o uso de CPU, 238-241  
    restauração de estado, 391-401  
    segurança para thread, 295

- segurança para threads, 294  
 universalizar, 249-250
- aplicativos universais  
 acomodar diferenças entre dispositivos, 369  
 acomodar diferenças nos dispositivos, 369  
 como acomodar diferenças entre dispositivos, 364  
 criar, 249-250  
 definição, 249  
 usar classes exclusivas do iPad, 365
- App ID, 20
- application:didFinishLaunchingWithOptions:**, 136
- application:shouldRestoreApplicationState:**, 392
- application:shouldSaveApplicationState:**, 392
- applicationDidBecomeActive:**, 309, 314
- applicationDidEnterBackground:**, 306, 309, 315
- applications,  
 (see also application bundle, debugging, projects, universal applications)
- applicationWillEnterForeground:**, 309, 314
- applicationWillResignActive:**, 309, 314
- ARC (Automatic Reference Counting),  
 (see also memory management)
- ARC (Contagem de referência automática)  
 benefícios de, 60-62  
 história de, 73  
 vs. contagem de referência manual, 73  
 e ciclos de referências fortes, 62-65  
 visão geral, 58
- archiveRootObject:toFile:**, 305
- área de depuração (Xcode), 32
- área de utilitários, 151
- área de utilitários (Xcode), 8
- área do editor (Xcode), 7
- área restrita do aplicativo, 303-305, 317
- área restrita, aplicativo, 303-305, 317
- áreas de trabalho (Xcode), 3
- argumentos, 26-27
- armazenamento de dados  
 para dados de aplicativos, 303-304  
 binários, 310, 315  
 escolher, 389  
 com funções de E/S, 315  
 para imagens, 326-328  
 com **NSData**, 309
- arquivamento  
 vs. Core Data, 371, 371  
 descrição, 301  
 imagens em miniatura, 328  
 com **NSKeyedArchiver**, 305-307  
 quando usar, 389  
 e arquivos XIB, 303
- arquivar  
 implementar, 301-303
- arquivo do modelo (Core Data), 371-376, 378
- arquivo do modelo de dados (Core Data), 371-376, 378
- arquivos  
 cabeçalho (.h), 35  
 copiar, 106  
 implementação (.m), 35, 36  
 importar, 36  
 incluir, 36
- arquivos .m, 35, 36
- arquivos de cabeçalho  
 vs. extensões de classe, 99  
 descrição, 35  
 importar, 36, 37-37, 55  
 ordem das declarações em, 47  
 pré-compilado, 55  
 visibilidade, 99
- arquivos de cabeçalho pré-compilados, 55
- arquivos de implementação, 35, 36
- arquivos NIB  
 e arquivamento, 303  
 arquivos XIB e, 11  
**awakeFromNib**, 344  
 carregar, 111  
 codificação de chave-valor e, 121  
 explicação, 11  
 trabalho de interface do usuário adicional, 344
- Arquivos NIB  
 carregamento manual, 322
- arquivos XIB  
 alternativa, 269  
 e arquivamento, 303  
 carregar manualmente, 158  
 conectar com arquivos-fonte, 172-176, 188-189, 321, 321, 321  
 conexões ruins em, 176  
 criar propriedades a partir de, 321, 321  
 definição, 7  
 editar, 7-15  
 fazer conexões em, 188-189  
 File's Owner, 112-114  
 fundamentos, 7  
 internacionalização de base e, 406  
 ~ipad e ~iphone, 269  
 localizar, 406-408  
 métricas simuladas, 174  
 versus arquivos NIB, 11  
 nomear, 118  
 placeholders em, 112  
 e propriedades, 321  
 quando usar, 109  
 vs. storyboards, 429
- arrays  
 acesso literal, 50, 50  
 conteúdo permitido, 49  
 criação literal, 48  
 definição, 29  
 vs. dicionários, 194  
 enumeração rápida de, 31, 32  
 gravação no sistema de arquivos, 317  
 e gerenciamento de memória, 60

- 
- e propriedade de objeto, 59, 60  
 princípios básicos de, 30, 31  
**atomic**, 69  
 atributos (Core Data), 372, 373-374  
 atributos de layout, 251, 252  
 atributos transformáveis (Core Data), 373, 374  
**Auto Layout**,  
     (see also constraints, Interface Builder)  
 atributos de layout, 251, 252  
 depuração, 262-268, 269  
 e Dynamic Type, 340  
 finalidade, 251  
 layout ambíguo, 263-265, 269  
 máscaras de redimensionamento automático e, 279, 280  
 restrições ausentes, 263-265, 269  
 restrições de placeholder, 346, 346  
 restrições insatisfatórias, 266  
 retângulos de alinhamento, 251, 252  
 visões posicionadas incorretamente, 266  
**\_autolayoutTrace**, 269  
**autorelease**, 73  
**availableMediaTypesForSourceType:**, 204  
 avisos da memória, 192  
 avisos de pouca memória, 192  
**awakeFromInsert**, 378  
**awakeFromNib**, 344
- B**
- backgroundColor** (**UIView**), 79, 88, 89  
 barra do depurador, 134  
**becomeFirstResponder**, 126  
 biblioteca de fragmentos de código, 151-153  
 biblioteca de objetos, 8, 9  
 bibliotecas  
     fragmentos de código, 151-153  
     objeto, 8  
 blocos, 329-330  
     animação e, 422-426  
     captura de variáveis, 332-334  
     de conclusão, 293-294  
 blocos de conclusão, 293-294  
**bounds**, 82  
 bundles  
     application (see application bundle)
- C**
- cadeia de respondentes, 216-216  
**CALayer**, 76  
 callbacks, 313 (see delegation, notifications, target-action pairs)  
 camadas (de visões), 76-77  
 camera,  
     (see also images)  
 câmera  
     gravar vídeo, 204-205  
     tirar fotos, 187-192
- caminhos de arquivo, recuperar, 304-305  
 caminhos de código, 242  
**cancelsTouchesInView**, 227  
**canPerformAction:withSender:**, 228  
 canvas (Interface Builder), 8  
 carregamento lento, 107, 119, 120  
 catálogos de ativos (Xcode), 21  
**cells** (see **UITableViewCell**)  
 certificados de desenvolvedor, 20  
**CGContextRef**, 93, 326  
**CGPoint**, 78  
**CGRect**, 78-80  
**CGRectMake()**, 79  
**CGSize**, 78, 326  
 chaves  
     declarações de variável de instância, 35, 35  
     sintaxe do dicionário, 196, 196  
 chaves (em dicionários), 194-200  
 ciclos de referências fortes, 62-65  
     encontrar usando o instrumento Leaks, 241-242
- class extensions,  
     (see also header files)
- classes,  
     (see also *individual class names*)  
     arquivos de cabeçalho de, 35-35  
     arquivos para, 35, 106  
     copiar arquivos, 141  
     criar, 6-7, 34-35  
     criar subclasses, 33-49, 100  
     herança de, 35  
     hierarquia de, 30, 30, 35  
     prefixos para, 54  
     reutilizar, 106, 141  
     subclasses, 30, 30  
     superclasses, 30, 30, 35, 43  
     visão geral, 25-26  
     visibilidade, 99  
 classes de restauração, 392  
**\_cmd**, 314  
 Cocoa Touch, 25  
 codificação de chave-valor (KVC), 121, 122  
 Código de amostra (documentação), 92  
 colchetes  
     arrays e, 50, 50, 50, 50  
     mensagens e, 26, 26  
     sintaxe de array, 48, 48  
 conexões (no Interface Builder), 11-15  
 configurações de idioma, 403, 408  
 configurações de região, 403  
 configurações do usuário, 415  
 console (Xcode), 32  
**constraintsWithVisualFormat:options:metrics:views:**, 273, 273  
**constraintWithItem:attribute:relatedBy:toItem:attribute:multiplier:constant:**, 277-279  
 contagem de referência manual, 73  
**contentMode** (**UIImageView**), 186-186  
**contentView** (**UITableViewCell**), 147-148, 319-320

- contentViewController (**UIPopoverController**)**, 285  
 contexto de gráficos, 93  
 contextos offscreen, 326-326  
 controladores de visão  
   adicionar a controlador de visão dividida, 363-365  
   adicionar ao controlador de navegação, 176-177  
   adicionar ao controlador popover, 286  
   apresentar, 114  
   carregamento lento de visões, 107, 119, 120  
   carregar visões, 112  
   criar em um storyboard, 429-447  
   definidos, 105  
   detalhes, 363  
   famílias de, 297, 298  
   mestre, 363, 366  
   modal, 191  
   papel no aplicativo, 105  
   passar dados entre, 177-178  
   recarregar subvisões, 192  
   relacionamentos entre, 297-300  
   e restauração de estado, 392-401  
   e hierarquia de visão, 106, 107  
 controladores de visão de detalhes, 363  
 controladores de visão mestre, 363, 366  
 controladores de visão modais  
   como dispensar, 290-291  
   definição, 191  
   estilos de, 291  
   e visão pai que não desaparece, 292  
   relacionamentos de, 298  
   em storyboards, 436-440  
   transições para, 294  
 controladores popover, 285-286, 366  
 convenções de nomenclatura  
   arquivos XIB, 118  
   identificadores de reutilização de célula, 150  
   métodos acessores, 36, 122  
   métodos inicializadores, 41  
   prefixos de classe, 54  
 copiar arquivos, 141  
 copiar objetos, 71-71  
 copy (atributo de propriedade), 71, 71  
 Core Data  
   vs. arquivamento, 371, 371  
   arquivo do modelo, 371-376, 378  
   atributos, 372, 373-374  
   busca tardia, 387  
   configurações de modelo, 389  
   controle de versão, 388, 389  
   criação de subclasses**NSManagedObject**, 376-378  
   entidades, 372-376, 382-386  
   **NSManagedObjectContext**, 378-382  
   **NSManagedObjectModel**, 378-379  
   **NSPersistentStoreCoordinator**, 378-379  
   objetos falsos, 387-388  
   como ORM, 371-372  
   propriedade obtida, 388  
   quando usar, 371, 389  
   registro de comandos SQL, 386  
   relacionamentos, 374-376, 387-388  
   solicitações de busca, 380-381, 388  
   e SQLite, 371, 378-379, 386  
   transformar valores, 373, 374  
 Core Graphics, 93-95  
 Core Graphics (framework), 326-326  
**count (NSArray)**, 31  
 credenciais (serviços web), 359-361  
 criação de subclasses  
   tipos de retorno do método, 42  
 criar perfis (aplicativos), 233-234  
 criar subclasses, 33-49, 100  
   uso de `self`, 48  
**currentLocale**, 404
- ## D
- dados JSON, 354-356  
 data storage,  
   (see also archiving, Core Data)  
**dataSource (**UITableView**)**, 139, 141-147, 146  
**dealloc**, 61  
 debugging,  
   (see also debugging tools, exceptions)  
 declarações  
   método, 36, 41-42, 47-48  
   propriedade, 66-69  
   protocolo, 130  
   variável de instância, 35  
 declarações antecipadas, 143  
**decodeRestorableStateWithCoder:**, 398  
**definesPresentationContext**, 299  
 delegação  
   protocolos usados para, 129-130  
   delegate de aplicativo, 18  
   delegates de aplicativos, 136  
**deleteRowsAtIndexPaths:withRowAnimation:**, 163  
 depuração  
   Auto Layout, 262-268, 269  
   exceções, 51-52  
**NSError**, 316-317  
 percorrer métodos, 134-135  
 rastreamento de pilha, 133  
 desafio de autenticação, 359  
 desalocação prematura, 59  
**description (NSObject)**, 33, 40  
 descritores de classificação (**NSFetchRequest**), 380  
 destinatário, 26  
 destinos  
   ajustes de compilação para, 244-247, 317  
   definição, 244  
 dicionários  
   criação literal (@{...}) e acesso, 196  
   gerenciamento de memória de, 313  
   gravação no sistema de arquivos, 317  
   usar, 194-196, 196-198  
 dictionaries,

---

(see also JSON data)

**didRotateFromInterfaceOrientation:**, 285

diretório Library/Preferences, 304

diretório tmp, 304

diretórios

- aplicativo, 303-305
- Documents, 304
- Library/Caches, 304
- Library/Preferences, 304
- lproj, 406, 412
- temporário, 304

diretórios lproj, 406, 412

**dismissPopoverAnimated:completion:**, 287

**dismissViewControllerAnimated:completion:**, 292, 293

**dispatch\_async()**, 357

**dispatch\_once()**, 294, 295

dispositivos

- implementar em, 20, 21
- provisionamento, 20
- provisionar, 20-21
- resolução de tela, 80
- tela Retina, 21, 23, 122-123
- verificar câmera, 190-191

dock (Interface Builder), 7

documentação do desenvolvedor, 85-86, 92, 302

documentação, desenvolvedor, 85-86, 92, 302

Documents diretório, 304

drawing (see views)

**drawRect:**, 82-92

- e loop de execução, 98, 99
- e **UITableViewCell**, 319

Dynamic Type (tipo dinâmico), 337-346

**E**

**editButtonItem**, 182

**editing (UITableView, UITableViewDelegate)**, 155, 160

editor assistente, 172-176, 188-189

efeitos de movimentação, 132, 132

**encodeInt:forKey:**, 302

**encodeObject:forKey:**, 302

**encodeRestorableStateWithCoder:**, 398, 399

**encodeWithCoder:**, 301, 301-302, 305

@end, 35

**endEditing:**, 179, 200

entidades (Core Data), 372-376, 382-386

enumeração rápida, 31, 31, 32, 42, 49

envios de mensagem aninhada, 26

erros

- e **NSError**, 316-317
- tempo de compilação, 51
- tempo de execução, 51-52

erros em tempo de compilação, 51

erros em tempo de execução, 51-52

esquemas, 19, 21

estado ativo, 308

estado de segundo plano, 308-309, 314-315

estado inativo, 308

estado segundo plano, 308

estado suspenso, 308, 309

estados do aplicativo, 307-309, 314-315

estados, aplicativo, 307-309

estilos de texto, 337

estruturas (C), 25-26

eventos

- callbacks e, 313
- controle, 217
- loop de execução e, 98, 99
- movimentação, 126
- primeiro respondente e, 126

eventos de controle, 217

eventos de movimentação, 126

eventos de toque

- ativar multitoque, 212-215
- princípios básicos, 207-208
- e cadeia de respondentes, 216-216
- e pares destino-ação, 216-217
- e **UIControl**, 216-217

**events**

- touch (see touch events)

exceção de inconsistência interna, 161

exceções

- diagnosticar no depurador, 135-135
- disparar, 289
- inconsistência interna, 161
- no Objective-C, 52
- ponto de interrupção para, 135-135
- seletor não reconhecido, 52
- uso **NSEException**, 289

exceptions

- explained, 51-52

**exerciseAmbiguousLayout (UIView)**, 264, 265

extensões de classe, 99, 100

**F**

ferramentas de depuração

- analisador estático, 242-243, 245
- análise de geração, 238, 238
- depurador, 132-135
- instrumento Allocations, 233-238
- Instrumentos, 233-242
- medidores de depuração, 231-233
- navegador de depuração, 133
- navegador de problemas, 19
- Pontos de interrupção, 132-133, 135
- rastreamento de pilha, 133, 133
- Time Profiler, 238-241
- visão de variáveis, 134, 134

File's Owner, 112-114, 112

fileiras (**UITableView**)

- adicionar, 161-162

**filteredArrayUsingPredicate:**, 381

**firstResponder**, 126

frame (em pilha), 57

**frame (UIView)**, 78-80, 82

frameworks  
 Core Data (see Core Data)  
 Core Graphics, 93-95, 326  
 Foundation, 54  
 importar, 55  
 MobileCoreServices, 205  
 prefixos e, 54  
 vinculação, 55  
 funções, 26  
 funções de E/S (I/O), 315  
 funções de tempo, 422-423  
 funções **UIGraphics**, 326-326  
 functions,  
 (see also *individual function names*)

**G**

**genstrings**, 410  
 gerenciamento de memória  
 com ARC, 58  
 arrays, 60  
 desalocação prematura, 59  
 dicionários, 196, 313  
 e instrumento Leaks, 241-242  
 necessidade de, 58  
 e propriedade de objeto, 59-62  
 otimizar com instrumento Allocations, 233-238  
 ponteiros, 59-62  
 ponteiros pendentes, 59  
 e propriedades, 70  
 referências fortes e fracas, 62-65  
 e ciclos de referências fortes, 62-65  
**UITableViewCell**, 150  
 vazamentos, 59, 62  
 gestos  
 arraste de tela, 100, 224, 225-227  
 toque longo, 224-225  
 toques curtos, 220-224  
 gestures,  
 (see also **UIGestureRecognizer**, **UIScrollView**)  
 gravação de vídeo, 204-205  
 Guias SDK (documentação), 92  
 GUIDs, 196

**H**

.h files (see header files)  
**hasAmbiguousLayout** (**UIView**), 264  
 heapshots, 238, 238  
 herança simples, 33, 35  
 herança, simples, 33, 35  
 hierarquia de visão, 107  
 hierarquia de visões, 76-82  
 hierarquias  
 classe, 30, 30, 35  
 visão, 76-82

**I**

**IBAction**, 14, 14, 188-189

**IBOutlet**, 12, 12, 174-176  
**ibtool**, 412-413  
 ícones  
 câmera, 187  
 catálogos de ativos para, 21  
 de aplicativo, 21-22  
 icons,  
 (see also images)  
**id**, 42  
 identificadores de restauração, 392-395, 392  
**ignoreSnapshotOnNextApplicationLaunch**, 401  
 image picker (see **UIImagePickerController**)  
**imageNamed:**, 123  
 imagens  
 armazenar, 193-196  
 armazenar em cache, 309-311  
 arquivamento, 326-328, 328  
 criar miniaturas, 326-328  
 exibir em **UIImageView**, 185-186  
 incluir na **NSData**, 310  
 manipular em contextos offscreen, 326-328  
 para tela Retina, 123  
 imagens de abertura, 22-23, 391, 402  
 imagens em miniatura, criar, 326-328  
**imagePickerController:**  
**didFinishPickingMediaWithInfo:**, 190  
**imagePickerControllerDidCancel:**, 190  
 images,  
 (see also camera, icons, **UIImageView**)  
**imageWithContentsOfFile:**, 311  
**@implementation**, 36  
**#import**, 37  
 importar arquivos, 36, 37, 55  
 incluir arquivos, 36  
 indicador acessório (**UITableViewCell**), 147  
 inicializadores, 41-45  
 designados, 41-45  
 proibir chamadas para, 289  
 e singletons, 142-145  
 inicializadores designados, 41-45  
**init**  
**alloc** e, 26-26  
 sobrescrever, 44  
 visão geral, 41-45  
**initialize**, 415  
**initWithCoder:**, 301, 303-303  
**initWithContentsOfFile:**, 315-317  
**initWithFrame:**, 78  
**initWithStyle:**, 140  
**insertObject:atIndex:**, 30, 50  
 inspetor (Xcode)  
 de atributos, 10  
 inspetor de arquivos, 407  
 inspetor de atributos, 10  
 inspetor de conexões, 15  
 inspetor de identidade, 113  
 inspetor de modelo de dados, 374  
 inspetor de tamanho, 277

- 
- inspetores (Xcode)  
    conexões, 15  
inspetores (Xcode)  
    arquivo, 407  
    encontrar, 8  
    identidade, 113  
    modelo de dados, 374  
    tamanho, 277  
instance variables,  
    (see also pointers, properties)  
**instancetype**, 42  
instâncias, 25-26  
instrumento Allocations, 233-238  
instrumento Leaks, 241-242  
Instrumento Time Profiler, 238-241  
Instrumentos, 233-242  
**integerForKey:**, 416  
**@interface**  
    em arquivos de cabeçalho, 35  
    em extensões de classe, 99  
Interface Builder  
    canvas, 7  
    conectar com arquivos-fonte, 172-176, 321, 321  
    conectar objetos, 11-15  
    configurar outlets em, 174, 175  
    criar objetos em, 8-11  
    definir destino-ação no, 14-15  
    definir outlets no, 12-13  
    dock, 7  
    editar arquivos XIB, 7-15  
    explicação, 7  
    fazer conexões em, 172-176  
    métricas simuladas, 174  
    placeholders no, 16, 17  
    e propriedades, 321, 321  
    quando usar, 109  
interface de usuário  
    drill-down, 167  
interface do usuário  
    drill-down, 363  
    lidar com rotação, 281, 283, 283, 285  
    orientação da, 281, 283, 283, 285  
    rolagem, 100-102  
    teclado, 200  
interface drill-down  
    com **UINavigationController**, 167  
    com **UISplitViewController**, 363  
interface files (see header files)  
**interfaceOrientation**, 285  
internacionalização, 403-405, 412  
internacionalização de base, 406  
internationalization,  
    (see also localization)  
iPad,  
    (see also devices)  
executar aplicativos de iPhone no, 2  
ícones de aplicativos para, 21  
imagens de abertura para, 23  
**isEqual:**, 163  
**isSourceTypeAvailable:**, 190  
itens da barra de guias, 116-118
- J**  
janela Organizer, 20
- K**  
**kUTTypeImage**, 204, 205  
**kUTTypeMovie**, 204, 205  
KVC (codificação de chave-valor), 121, 122
- L**  
layout ambíguo, 263-265, 269  
libraries,  
    (see also frameworks)  
Library/Caches diretório, 304  
linhas (**UITableView**)  
    excluir, 162-163  
    mover, 163-164  
linhas de base, 252  
lista de projetos e destinos (Xcode), 244  
listas de propriedades XML, 317  
**loadView**, 107, 107, 107, 159  
**Localizable.strings**, 410  
localização  
    arquivos XIB, 406-408  
    configurações do usuário para, 403, 408  
    diretórios `lproj`, 412  
    usar `ibtool`, 412-413  
    internacionalização, 403-405, 412  
    internacionalização de base e, 406  
    **NSBundle**, 412  
    de preferências, 420  
    recursos, 406-408  
    tabelas de strings, 409-411  
localizar  
    diretórios `lproj`, 406  
**localizedDescription**, 316  
**locationInView:**, 223  
loop de evento, 98, 99  
loop de execução, 98, 99, 136
- M**  
arquivos `.m`, 35, 36  
**mach\_msg\_trap()**, 240  
macros de pré-processador, 246-247  
macros, pré-processador, 246-247  
main bundle (see application bundle)  
**main()**, 30, 136  
**main.m**, 30  
**mainBundle**, 158, 318  
Mapeamento objeto-relacional (ORM), 371  
máscaras de bits, 295-296  
máscaras de redimensionamento automático, 271, 279, 280  
**mediaTypes**, 204

medidores de depuração, 231-233  
 memória, 57, 57, 58  
 memória heap, 57, 58  
 mensagens, 26-28  
**menus (`UIMenuController`)**, 223-224, 228  
 messages,  
   (see also methods)  
 methods,  
   (see also *individual method names*)  
 métodos  
   ação, 216-217  
   acessores, 36-40  
   classe, 40, 47-48  
   conveniência, 48  
   de ação, 14-15  
   de fonte de dados, 146  
   declarar, 36, 41, 41-42, 47  
   definidos, 26  
   vs. funções, 26  
   implementar, 36-37, 41, 42  
   inicializador, 41-45  
   inicializador designado, 41-45  
   instância, 40  
   vs. mensagens, 27  
   nomes de, 27  
   percorrer, 134-135  
   sobrescrever, 40-41, 43-45  
 métodos acessores, 36-40  
   convenções de nomenclatura para, 122  
   importância de usar, 40  
   personalizar, 71, 72  
   propriedades e, 66-69, 145  
   sintaxe de ponto e, 39-40  
 métodos de ação, 14-15  
   conectar em arquivo XIB, 188-189  
   e **UIControl**, 216-217  
 métodos de classe, 40, 47-48  
 métodos de conveniência, 48  
 métodos de fonte de dados, 146  
 métodos de instância, 40  
 métodos getter, 36-37  
 métodos obrigatórios (em protocolos), 130  
 métodos opcionais (em protocolos), 130  
 métodos setter, 36-37  
 métricas simuladas, 174  
**minimumPressDuration**, 225  
**MobileCoreServices**, 205  
 arquivos `.mobileprovision`, 20-20  
**modalPresentationStyle**, 291, 299  
**modalTransitionStyle**, 294  
**modalViewController**, 290-291  
**Modelo-Visão-Controlador (MVC)**, 4-5, 314  
**Modelo-Visão-Controlador-Armazenamento (MVCS)**, 314  
 multi-thread, 357  
**multipleTouchEnabled (`UIView`)**, 212  
 multisegmentação, 294, 295  
 multitoque, ativar, 212

**MVC (Modelo-Visão-Controlador)**, 4-5, 314  
**MVCS (Modelo-Visão-Controlador-Armazenamento)**, 314

## N

namespaces, 54  
 navegador de depuração, 133  
 navegador de ponto de interrupção, 135  
 navegador de problemas, 19  
 navegador de projetos, 3  
 navegador do projeto, 3-4  
 navegadores (Xcode)  
   de projetos, 3  
   definição, 3  
   depurar, 133  
   Pontos de interrupção, 135  
   problemas, 19  
   projeto, 3-3  
 navigation controllers (see **UINavigationController**)  
**navigationController**, 176, 298  
**navigationItem (`UIViewController`)**, 179  
**nextResponder**, 216  
 NIB files,  
   (see also XIB files)  
**nibWithNibName:bundle:**, 322  
**nil**  
   e arrays, 49  
   definir ponteiros, 27  
   enviando mensagens para, 28  
   retornado do inicializador, 44  
   ações de destino, 217  
   como ponteiro nulo, 27  
**nonatomic**, 69  
 notificações (**NSNotificationCenter**), 311-314  
   configurar alteração, 420  
   de avisos de pouca memória, 311  
   notificações locais, 119, 119  
   notificações push, 119  
   notificações, locais, 119, 119  
   notificações, push, 119  
**NSArray**,  
   (see also arrays)  
**count**, 31  
   criação literal (@[...]), 48  
   detalhes, 49-50  
**objectAtIndex:**, 50  
   princípios básicos, 30, 31  
**NSBundle**, 158, 412  
**NSCoder**, 301, 303  
   para restauração de estado, 398, 399  
**NSData**, 49, 309, 328, 415  
**NSDate**, 178, 317  
**NSDateFormatter**, 178, 404  
**NSDictionary**, , 194-196  
   (see also dictionaries)  
**NSError**, 316-317  
**NSException**, 289  
**NSExpression**, 388

---

**NSFetchRequest**, 380-381, 388  
**NSGlobalDomain**, 417  
**NSIndexPath**, 149, 163  
**NSInteger**, 147  
**NSJSONSerialization**, 355  
**NSKeyedArchiver**, 305-307  
**NSKeyedUnarchiver**, 306  
**NSLayoutConstraint**, 273  
**NSLocale**, 404  
**NSLocalizedString**, 411  
**NSLocalizedString()**, 409  
**NSLog()**, 32  
**NSManagedObject**, 376-378, 389  
**NSManagedObjectContext**, 378-382, 389  
**NSManagedObjectModel**, 378-379  
**NSMutableArray**, , 30  
    (see also arrays)  
    detalhes, 49-50  
    **insertObject:atIndex:**, 50  
    princípios básicos, 30  
    **removeObject:**, 163  
    **removeObjectIdenticalTo:**, 163  
    **replaceObjectAtIndex:withObject:**, 50  
**NSMutableDictionary**, , 193-196  
    (see also dictionaries)  
**NSNotificationCenter**, 311-314  
**NSNull**, 49  
**NSNumber**, 49, 49, 317  
**NSObject**, 30, 30, 33-35  
    **dealloc**, 61  
    **description**, 33, 40  
**NSPersistentStoreCoordinator**, 378-379  
**NSPredicate**, 381  
**NSSearchPathDirectory**, 304  
**NSSearchPathForDirectoriesInDomains**, 304  
**NSSortOrdering**, 388  
**NSString**  
    criação literal (@"..." ), 31  
    criar, 31, 48  
    gravar no sistema de arquivos, 310-315  
    internacionalização, 409  
    localizar, 412-413  
    **NSLog()** e, 32  
    princípios básicos, 30, 31  
    serializável com lista de propriedades, 317  
    **stringWithFormat:**, 48  
    usar tokens com, 32  
**NSStringFromSelector**, 314  
**NSTemporaryDirectory**, 304  
**NSUInteger**, 147  
**NSURL**, 352-353  
**NSURLCredential**, 360  
**NSURLRequest**, 352-353, 361-362  
**NSURLSession**, 352, 353-354, 359  
**NSURLSessionAuthChallengeUseCredential**, 360  
**NSURLSessionConfiguration**, 353  
**NSURLSessionDataDelegate** (protocolo), 360  
**NSURLSessionDataTask**, 353-354, 355, 356, 357  
**NSURLSessionTask**, 352, 361  
**NSUserDefaults**, 304, 415  
**NSUserDefaultsDidChangeNotification**, 420  
**NSUUID**, 196-198  
**NSValue**, 49  
**NSValueTransformer**, 373, 374  
números binários, 296, 297  
números de linha, exibir, 243

**O**

**objc\_msgSend()**, 240  
**objectAtIndex: (NSArray)**, 50  
**objectForKey:**, 194-196  
Objective-C  
    convenções de nomenclatura, 36, 41  
    herança simples em, 35  
    nomes de mensagens, 27, 27  
    nomes de métodos, 27  
    palavras-chave, 35  
    prefixo @ , 35  
    princípios básicos, 25-52  
objects,  
    (see also classes, memory management)  
objetos  
    alocação, 57  
    copiar, 71-71  
    independência de, 36  
    na memória, 57  
    propriedade de, 58-62  
    serializável com lista de propriedades, 317  
    tamanho de, 57  
    visão geral, 25-26  
objetos de armazenamento, 314  
objetos de controlador, 4, 314  
objetos de modelo, 4  
objetos de placeholder, 112  
objetos falsos, 387-388  
objetos serializáveis com lista de propriedades, 317  
operadores binários, 297  
orientação da interface, 281, 283, 283, 285  
orientação do dispositivo, 281, 283, 283, 285  
**orientation (UIDevice)**, 281  
ORM (Mapeamento objeto-relacional), 371  
outlets  
    conectar com arquivos-fonte, 321, 321  
    configurar, 172  
    declaração como weak (fraco), 114  
    definição, 11  
    definir, 12-13

**P**

pacote de aplicativo, 318  
    explicação, 317-318  
    e internacionalização, 412  
pacote de configurações, 415, 417-420  
pacote do aplicativo  
     **mainBundle**, 158

**NSBundle**, 158  
pacote principal, 158  
pacotes  
  configurações, 415, 417-420  
  identificadores para, 20  
**NSBundle**, 317, 412  
páginas de referência, 85-86, 302  
parallax, 132  
parentViewController, 290-291  
pares chave-valor, 194-196  
pares destino-ação  
  configurar de forma programática, 181  
  definição, 14-15  
  e **UIControl**, 216-217  
  e **UIGestureRecognizer**, 219  
**pathForResource:type:**, 412  
perfis de provisionamento, 20-20  
pilha (memória), 57, 57, 133  
pixels, 80  
placeholders (no código), 17  
placeholders (no código), 16, 152  
pointers  
  in Interface Builder (see outlets)  
ponteiro **isa**, 50  
ponteiros  
  em arrays, 30  
  definir em arquivos XIB, 12-13  
  definir **nil**, 27  
  e gerenciamento de memória, 59-62  
  sintaxe de, 35  
  como referências fortes, 62  
  visão geral, 26-26  
  como referências fracas, 62, 62, 64, 65, 65  
ponteiros pendentes, 59  
Pontos (vs. pixels), 80  
Pontos de interrupção, 132-133, 135  
pool de liberação automática, 73  
**popoverControllerDidDismissPopover:**, 286  
**#pragma mark**, 203, 203  
predicados (solicitações de busca), 381  
**predicateWithFormat:**, 381  
Preenchimento automático (no Xcode), 16, 17, 151-153  
Preenchimento de código (no Xcode), 16, 17, 151-153  
preferences,  
  (see also Dynamic Type, localization)  
preferências  
  atualizar, 417-420  
  constantes globais para, 416  
  fonte, 337-340, 342, 343  
  ler, 416  
  localização de, 415  
  localizar, 420  
  pacote de configurações e, 417-420  
  padrões disponíveis, 417  
  registrar padrões, 416  
preferências de fonte, 337-340, 342, 343  
**preferredContentSizeCategory (UIApplication)**, 342  
**preferredFontForTextStyle: (UIFont)**, 338  
prefixo @  
  criar arrays com, 48  
  criar strings com, 31  
  Palavras-chave do Objective-C, 35  
**prepareForSegue:sender:**, 445  
presentedViewController, 298  
presentingViewController, 290, 291, 298  
**presentViewController:animated:completion:**, 191, 293  
primeiro respondente  
  campos de texto e, 126  
  desistir, 200  
  e ações de destino **nil**, 217  
  para eventos sem toque, 126  
  renunciar, 179  
  e cadeia de respondentes, 216-216  
  tornar-se, 126  
  e **UIMenuController**, 224  
prioridade de envolvimento de conteúdo, 276  
prioridade de resistência à compressão de conteúdo, 276  
produtos, 244  
projetos  
  ajustes de compilação para, 244-247  
  configurações de destino em, 317  
  copiar arquivos para, 141  
  criar, 1-3  
  criar novos, 3  
  definição, 243  
  limpar e construir, 408  
  templates para, 2  
propriedade  
  declarar, 66-69  
propriedade obtida, 388  
propriedades  
  acessores personalizados para, 71, 72  
  atomic, 69  
  atributos de, 69-71  
  copy, 71, 71  
  criar a partir de um arquivo XIB, 321, 321  
  criar através do Interface Builder, 321  
  criar no Interface Builder, 321  
  gerenciamento de memória de, 70  
  sem variáveis de instância, 72  
  métodos acessores, 145  
  nonatomic, 69  
  readonly, 70  
  readwrite, 70  
  sintetizar, 72, 145  
  sobrescrever acessores, 72  
  strong, 70  
  visibilidade, 99  
  weak, 70  
protocolo HTTP, 361-362  
protocolo NSCoding, 301-303

---

protocolos  
declarar, 130  
delegate, 129-130  
descrição, 129-130  
estrutura de, 130  
métodos opcionais vs. obrigatórios, 130  
**NSCoding**, 301-303  
**NSURLSessionDataDelegate**, 360  
**UIApplicationDelegate**, 309  
**UIDataSourceModelAssociation**, 400  
**UIGestureRecognizerDelegate**, 226  
**UIImagePickerControllerDelegate**, 190, 192-192  
**UINavigationControllerDelegate**, 192  
**UIPopoverControllerDelegate**, 286  
**UIResponderStandardEditActions**, 228  
**UISplitViewControllerDelegate**, 367  
**UITableViewDataSource**, 139, 146-147, 148, 163, 163  
**UITableViewDelegate**, 139  
**UITextFieldDelegate**, 129, 200  
**UIViewControllerRestoration**, 396  
**pushViewController:animated:**, 176-177

## Q

Quartz (see Core Graphics)  
Quick Help, 302

## R

rastreamento de pilha, 133, 133  
**readonly**, 70  
**readwrite**, 70  
recursos  
catálogos de ativos para, 21  
definição, 21, 317  
localizar, 406-408  
Referência de API, 85-86, 302  
referências fracas, 62, 64, 65, 65, 70  
relacionamentos (Core Data), 374-376, 387-388  
**release**, 73  
**removeObject:**, 163  
**removeObjectIdenticalTo:**, 163  
reordenar controles, 164  
**replaceObjectAtIndex:withObject:**, 50  
**requireGestureRecognizerToFail:**, 228  
**resignFirstResponder**, 126, 131, 179  
resolução de tela, 80  
responders (see first responder, **UIResponder**)  
**respondsToSelector:**, 130  
restauração de estado  
e ciclo de vida do aplicativo, 395-396  
controle de instantâneos, 401  
explicação, 391-392  
identificadores de restauração, 392, 393-395  
e **NSCoder**, 398  
optar pela, 392  
optar por, 392  
com storyboards, 447

**UIViewControllerAnimatedRestoration** protocolo, 396  
para visões, 399  
restrições (Auto Layout)  
adição de forma programática, 274-275  
alinhamento, 259-260  
ausência, 263-265  
ausentes, 269  
criação sem VFL, 277-279  
criar de forma programática, 272-279  
criar no Interface Builder, 253-262  
depuração, 262-268, 269  
exclusão, 257  
fixação, 254-258  
insatisfatórias, 266  
placeholder, 346, 346  
prioridades, 262, 276  
tamanho de conteúdo intrínseco, 276-277  
vizinho mais próximo e, 253  
restrições (Layout Automático)  
criação com VFL, 272-273  
visão geral, 252  
restrições ausentes, 263-265, 269  
restrições insatisfatórias, 266  
**retain**, 73  
retângulos de alinhamento, 251, 252  
**reuseIdentifier** (**UITableViewCell**), 150  
reutilização  
células da visão de tabela, 150-151  
reutilizar  
classes, 141  
rolagem, 100-102  
**Root.plist**, 419-420  
**rootViewController** (**UINavigationController**), 168-170  
**rootViewController** (**UIWindow**), 108, 108  
rotação automática, 281, 283, 283, 285  
rotação, lidar com, 281, 283, 283, 285  
rótulos (em nomes de mensagens), 27

## S

seções (**UITableView**), 147, 155-155  
**SEL**, 181  
**@selector()**, 181  
seletor, 26, 27  
seletor não reconhecido, 52  
**self**, 43-43, 48  
**sendAction:to:from:forEvent:**, 217  
**sendActionsForControlEvents:**, 217  
serviços web  
autenticação, 359-361  
credenciais, 359-361  
para armazenamento de dados, 389  
e protocolo HTTP, 361-362  
implementar, 352-357  
com dados JSON, 354-356  
**NSURLSession**, 353-354  
solicitar dados de, 352-354  
SSL (Secure Sockets Layer), 359, 360

- visão geral, 350-350  
**setEditing:animated:**, 160, 182  
**setNeedsDisplay (UIView)**, 99  
**setNeedsDisplayInRect: (UIView)**, 99  
**setObject:ForKey:**, 194-196  
**setPagingEnabled:**, 102  
**setStroke (UIColor)**, 88  
 settings (see preferences)  
**setValue:ForKey:**, 121  
 simulador  
   aviso de pouca memória e, 313  
   desativar aplicativos no, 391  
   executar aplicativos no, 19  
   exibição do pacote de aplicativo no, 317  
   girar no, 282  
   local da área restrita, 306  
   múltiplos toques em, 136  
   salvar imagens em, 191  
 simulador de iOS  
   aviso de pouca memória e, 313  
   desativar aplicativos no, 391  
   exibição do pacote de aplicativo no, 317  
   girar no, 282  
   múltiplos toques em, 136  
   salvar imagens em, 191  
 simulador do iOS  
   local da área restrita, 306  
 simulador iOS  
   executar aplicativos no, 19  
 simultaneidade, 357  
 singletons  
   implementar, 142-145  
   seguros para thread, 295  
   seguros para threads, 294  
 sintaxe de ponto, 39-40  
 sobrescrevendo métodos, 40-41  
 sobrescrever métodos, 43-45  
 solicitações de busca, 380-381, 388  
**sourceType (UIImagePickerController)**, 189-190  
 split view controllers (see **UISplitViewController**)  
**splitViewController**, 298, 365  
 SQL, 386  
 SQLite, 371, 378-379, 386  
 SSL (Secure Sockets Layer), 359, 360  
**standardUserDefaults**, 415  
 storyboards  
   criar, 429-432  
   prós e contras, 447-447  
   restauração de estado e, 447  
   segue, 435-446  
   tabelas estáticas em, 432-435  
   vs. arquivos XIB, 429  
 strings (see **NSString**)  
 strings de formatação, 32  
**stringWithFormat:**, 48  
 strong, 70  
 subclasses, 30, 30  
 subclassing,
- (see also overriding methods)  
 subvisões, 76  
 super, 43  
 superclasses, 30, 30, 35, 43  
 superview, 80  
**supportedInterfaceOrientations**, 283
- ## T
- tab bar controllers (see **UITabBarController**)  
**tabBarController**, 298  
 tabelas (banco de dados), 371  
 tabelas de strings, 409-411  
 tabelas estáticas, 432-435  
 table view cells (see **UITableViewCell**)  
 table view controllers (see **UITableViewcontroller**)  
 table views (see **UITableView**)  
**tableView**, 162  
**tableView:cellForRowAtIndexPath:**, 146, 148-150  
**tableView:commitEditingStyle:**  
**forRowAtIndexPath:**, 163  
**tableView:didSelectRowAtIndexPath:**, 178  
**tableView:moveRowAtIndexPath:toIndexPath:**, 163, 164  
**tableView:numberOfRowsInSection:**, 146-147  
 tamanho de conteúdo intrínseco, 276-277  
 teclado  
   aparência, 126  
   dispensar, 200  
   teclado numérico, 183  
   teclado numérico, 183  
 tela Retina, 21, 23, 122-123  
 templates (Xcode), xvi, 2  
**textFieldShouldReturn:**, 128, 200  
 thread principal, 357  
 thread UI, 357  
 threads, 294, 295, 357  
 tipo de dispositivo  
   configuração, 249  
   determinar no tempo de execução, 283  
**toggleEditMode:**, 160  
 token %@, 33  
 tokens, 32, 33  
**topViewController (UINavigationController)**, 168  
 touch events,  
   (see also **UIGestureRecognizer**)  
**touchesBegan:**  
**withEvent:**, 207, 208  
**touchesCancelled:**  
**withEvent:**, 207  
**touchesEnded:**  
**withEvent:**, 207, 208  
**touchesMoved:**  
**withEvent:**, 207, 208  
**translationInView:**, 226
- ## U
- UIAlertView**, 316

---

**UIApplication**  
e eventos, 208  
e `main()`, 136  
e cadeia de respondentes, 216, 217  
**UIApplicationDelegate**, 309  
**UIApplicationDidBecomeActiveNotification**, 401  
**UIApplicationDidReceiveMemoryWarningNotification**, 311  
**UIApplicationWillResignActiveNotification**, 401  
**UIBarButtonItem**, 180-182, 187-189, 201  
**UIBezierPath**, 84-92  
**UICollectionView**, 334-335  
**UIColor**, 79, 88, 88  
**UIContentSizeCategoryDidChangeNotification**, 340  
**UIControl**, 200-201, 216-217  
**UIControlEventTouchUpInside**, 217, 217  
**UIDataSourceModelAssociation** (protocolo), 400  
**UIFont**, 338  
**UIGestureRecognizer**  
adiar toques, 228  
arraste de tela, 224, 225-227  
`cancelsTouchesInView`, 227  
criar subclasses, 229  
descrição, 219  
detectar toques curtos, 220-224  
encadear reconhecedores, 228  
habilitar reconhecedores simultâneos, 226  
implementar múltiplos, 221-223, 225-227  
interceptar toques da visão, 227  
interceptar toques em uma visão, 226  
interceptar toques na visão, 220  
`locationInView`: , 223  
mensagens de ação da, 225  
mensagens de ação de, 219  
`state` (propriedade), 225, 226, 228  
subclasses, 219, 229  
toque longo, 224-225  
`translationInView`: , 226  
e **UIResponder** métodos, 227  
**UIGestureRecognizerDelegate**, 226  
**UIGraphicsBeginImageContextWithOptions**, 326  
**UIGraphicsEndImageContext**, 326  
**UIGraphicsGetImageFromCurrentImageContext**, 326  
**UIImage**, , 310  
(see also images, **UIImageView**)  
**UIImageJPEGRepresentation**, 310  
**UIImagePickerController**  
apresentar, 191-192  
criar instâncias, 189-190  
gravar vídeo com, 204-205  
no iPad, 285  
em **UIPopoverController**, 285  
**UIImagePickerControllerDelegate**, 190, 192-192  
**UIImageView**  
descrição, 185-186  
**UIInterpolatingMotionEffect**, 422  
**UILocalNotification**, 119  
**UILongPressGestureRecognizer**, 224-225  
**UIMenuController**, 223-224, 228  
**UIModalPresentationCurrentContext**, 299  
**UIModalPresentationFormSheet**, 291  
**UIModalPresentationPageSheet**, 291  
**UIModalTransitionStyleCoverVertical**, 294  
**UIModalTransitionStyleCrossDissolve**, 294  
**UIModalTransitionStyleFlipHorizontal**, 294  
**UIModalTransitionStylePartialCurl**, 294  
**UINavigationBar**, 168, 170-182  
**UINavigationController**,  
(see also view controllers)  
adicionar controladores de visão a, 176-177, 178  
criar instâncias, 170  
descrição, 168-171  
gerenciar pilha de controlador de visão, 168  
`navigationController`, 298  
**pushViewController:animated:**, 176-177  
`rootViewController`, 168-169  
em storyboards, 435, 436  
`topViewController`, 168-169  
e **UINavigationBar**, 179-182  
`view`, 168  
`viewControllers`, 168  
**viewWillAppear:**, 178  
**viewWillDisappear:**, 178  
**UINavigationControllerDelegate**, 192  
**UINavigationItem**, 179-182  
**UINib**, 322  
**UIPanGestureRecognizer**, 224, 225-227  
**UIPopoverController**, 285-286, 366  
**UIPopoverControllerDelegate**, 286  
**UIResponder**, 126  
ações de menu, 228  
e cadeia de respondentes, 216-216  
e eventos de toque, 207  
**UIResponderStandardEditActions** (protocolo), 228  
**UIScrollView**  
rolagem, 100-102  
**UISplitViewController**  
controladores de visão mestre e de detalhe, 363-366  
illegal no iPhone, 364  
no modo retrato, 366-368  
`splitViewController`, 298  
visão geral, 363-365  
**UISplitViewControllerDelegate**, 367  
**UIStoryboard**, 429-447  
**UIStoryboardSegue**, 435-446  
**UITabBarController**, 114-118, 167, 168  
`tabBarController`, 298  
`view`, 115  
**UITabBarItem**, 116-118  
**UITableView**, , 137-139  
(see also **UITableViewCell**, **UITableViewcontroller**)  
adicionar linhas a, 161-162  
editar modo de, 182

excluir linhas de, 162-163  
 modo de edição de, 155, 160-160, 320  
 mover linhas em, 163-164  
 preenchimento, 141-149  
 propriedade editing, 155, 160-160  
 seções, 147, 155-155  
 view, 141  
 visão de cabeçalho, 155-159  
 visão de rodapé, 155

**UITableViewCell**

- adicionar imagens a, 326-328
- contentView, 147-148, 319-320
- criação de subclasses, 319-323
- criar interface com um arquivo XIB, 320-322
- editar estilos, 163
- estilos de células, 148
- recuperar instâncias de, 148-149
- retransmitir ações de, 328-332
- reutilização de instâncias de, 150-151
- subvisões, 147-148
- UITableViewCellStyle, 148

**UITableViewCellEditingStyleDelete**, 163

**UITableViewController**,  
(see also **UITableView**)

- adicionar linhas, 161-162
- criar no storyboard, 432-435
- criar subclasse, 139-141
- criar tabelas estáticas, 432-435
- dataSource, 141-147
- descrição, 139-139
- excluir linhas, 162-163
- initializador designado, 140
- initWithStyle:**, 140
- métodos de fonte de dados, 146
- mover linhas, 163-164
- propriedade editing, 160
- retornar células, 148-151
- tableView**, 162
- UITableViewStyleGrouped, 140
- UITableViewStylePlain, 140

**UITableViewDataSource** (protocolo), 139, 146-147, 148, 163, 163

**UITableViewDelegate**, 139

**UITapGestureRecognizer**, 220-224

**UITextField**

- definir atributos de, 183
- como primeiro respondente, 200, 217
- UITextFieldTraits**, 126, 127

**UITextFieldDelegate**, 129, 200

**UITextFieldTraits (**UITextField**)**, 126, 127

**UIToolbar**, 180, 187

**UITouch**, 208-208, 211, 212, 212-215

**UIUserInterfaceIdiomPad**, 283

**UIUserInterfaceIdiomPhone**, 283

**UIView**,  
(see also **UIViewController**, views)

- backgroundColor**, 79, 88, 89
- bounds**, 82
- criar instâncias, 78
- criar subclasse, 77-82
- definida, 75-76
- drawRect:**, 82-92, 98, 99
- endEditing:**, 179
- exerciseAmbiguousLayout**, 264, 265
- frame, 78-80, 82
- hasAmbiguousLayout**, 264
- setNeedsDisplay**, 99
- setNeedsDisplayInRect:**, 99
- superview, 80

**UIViewController**,  
(see also view controllers)

- criar instância, xvi
- definesPresentationContext**, 299
- didRotateFromInterfaceOrientation:**, 285
- interfaceOrientation**, 285
- loadView**, 107, 107, 107, 159
- modalTransitionStyle**, 294
- modalViewController**, 290-291
- navigationController**, 176
- navigationItem**, 179
- parentViewController**, 290-291
- presentingViewController**, 290, 291
- splitViewController**, 365
- supportedInterfaceOrientations**, 283
- tabBarItem**, 116
- view**, 106, 112, 120, 216
- viewControllers**, 297
- viewDidLayoutSubviews**, 264
- viewDidLoad**, 120
- viewWillAppear:**, 120, 192
- willAnimateRotationToInterfaceOrientation:**  
**duration:**, 284
- e arquivos XIB, xvi

**UIViewControllerAnimatedRestoration** (protocolo), 396

**UIWebView**, 358-359

**UIWindow**, 76

- e cadeia de respondentes, 216
- rootViewController**, 108, 108

**unarchiveObjectWithFile:**, 306

**URLs**, , 352  
(see also **NSURL**)

user interface,  
(see also Auto Layout, views)

user settings (see preferences)

**userInterfaceIdiom**, 283

UUIDs, 196

**V**

**valueForKey:**, 121

variables,  
(see also instance variables, local variables, pointers, properties)

variáveis

- estáticas, 143
- local, 57, 58

variáveis de instância

---

declarar, 35-35  
descrição, 25  
explicação, 35-36  
na memória, 57  
métodos acessores para, 36  
visibilidade, 99  
e referências fracas, 65  
variáveis estáticas, 143  
variáveis locais, 57, 58  
vazamentos, memória, 59, 62  
VFL (Visual Formal Language), 272-273  
**view (**UIViewController**)**, 106, 120  
view controllers,  
    (see also **UIViewController**, views)  
**viewControllers**, 297  
**viewControllers (**UINavigationController**)**, 168  
**viewControllerWithRestorationIdentifierPath:coder:**  
**viewControllerWithRestorationIdentifierPath:coder:**, 396  
**viewDidLayoutSubviews (**UIViewController**)**, 264  
**viewDidLoad**, 120  
views,  
    (see also Auto Layout, touch events, **UIView**, view controllers)  
**viewWillAppear:**, 120, 178-179, 178, 192  
**viewWillDisappear:**, 178  
visão de cabeçalho (**UITableView**), 155-159  
visão de variáveis, 134, 134  
visibilidade, 99  
visões  
    adicionar à janela, 76, 107  
    animação, 421-426  
    apresentação modal de, 191  
    camadas e, 76-77  
    carregamento lento de, 107, 119  
    carregar, 112  
    criar personalizado, 77-82  
    definida, 75-76  
    em hierarquia, 76-77  
    em Modelo-Visão-Controlador, 4  
    projetar formas, 84-92  
    projetar na tela, 76-77, 82, 82, 98, 99  
    redesenhar, 98, 99  
    redimensionar, 186-186  
    renderizar, 76-77, 82, 82, 98, 99  
    rolagem, 100-102  
    e loop de execução, 98, 99  
    e restauração de estado, 399  
    e subvisões, 76-82  
    tamanho e posição de, 78-80, 82  
visões posicionadas incorretamente, 266  
Visual Formal Language (VFL), 272-273  
vizinho mais próximo, 253

## W

weak, 70  
**willAnimateRotationToInterfaceOrientation:duration:**, 284

**writeToFile:atomically:**, 310  
**writeToFile:atomically:error:**, 315

## X

.xcassets (catálogo de ativos), 21  
.xcdatamodeld (arquivo do modelo de dados), 372  
Xcode,  
    (see also debugging tools, inspectors, Instruments, libraries, navigators, projects, simulator)  
ajustes de compilação, 244-247  
analisador estático, 242-243, 245  
área de depuração, 32  
área de navegação, 3  
área de utilitários, 8, 151  
área do editor, 7  
áreas de trabalho, 3  
atalhos de teclado, 176  
biblioteca, 8  
biblioteca de fragmentos de código, 151-153  
biblioteca de objetos, 8, 9  
canvas, 8  
catálogos de ativos, 21  
console, 32  
construir interfaces, 7-15  
criar classes, 6-7  
criar perfis de aplicativos em, 233-234  
criar projetos em, 1-3  
depurador, 132-135  
destinos, 244  
editor assistente, 172-176, 188-189  
esquemas, 19, 21  
guias, 176  
inspetor de arquivos, 407  
inspetor de atributos, 10  
inspetor de identidade, 113  
inspetor de modelo de dados, 374  
inspetor de tamanho, 277  
inspetores, 8  
janela Organizer, 20  
lista de projetos e destinos, 244  
medidores de depuração, 231-233  
navegador da documentação, 85-86  
navegador de problemas, 19  
navegador de projetos, 3  
navegadores, 3  
nímeros de linha em, 243  
placeholders no, 152  
preenchimento de código, 16, 17, 151-153  
produtos, 244  
projetos, 243  
Quick Help, 302  
Referência de API, 85-86, 302  
templates, 78  
templates de aplicativo, 2  
versões, 2  
XIB files,  
    (see also Interface Builder, NIB files)

