

spis treści

wprowadzenie xvii

podziękowania xix

o książce xxi

CZĘŚĆ I PODSTAWOWE POJĘCIA 1

1 Wprowadzenie do programowania funkcyjnego 3

1.1 Czym jest to, co nazywamy programowaniem funkcyjnym? 4

Funkcje jako elementy pierwszoklasowe 4 ■ Unikanie zmiany stanu 5 ■ Pisanie programów o silnych gwarancjach 6

1.2 W jakim stopniu język C# jest funkcyjny? 9

Funkcyjna natura LINQ 10 ■ Elementy funkcyjne w C# 6 i C# 7 11 ■ Bardziej funkcyjna przyszłość dla języka C#? 13

1.3 Myślenie w kategoriach funkcji 14

Funkcje jako odwzorowania 14 ■ Reprezentacja funkcji w C# 15

1.4 Funkcje wyższego rzędu 19

Funkcje zależne od innych funkcji 20 ■ Funkcje adapterowe 22 ■ Funkcje, które tworzą inne funkcje 22

1.5 Używanie funkcji HOF do unikania duplikacji 23

Enkapsulacja przygotowania i zwolnienia w HOF 25 ■ Włączanie instrukcji using do HOF 26 ■ Kompromisy HOF 28

1.6 Korzyści wynikające z programowania funkcyjnego 29

2 Dlaczego czystość funkcji ma znaczenie 32

2.1 Czym jest czystość funkcji? 33

Czystość i efekty uboczne 33 ■ Strategie zarządzania efektami ubocznymi 34

2.2 Czystość i współbieżność 37

Czyste funkcje są dobre do równoległego przetwarzania 38 ■ Równoległe wykonywanie funkcji nieczystych 39 ■ Unikanie zmiany stanu 40

2.3 Czystość i testowalność 42

W praktyce: scenariusz walidacji 43 ■ Testowanie funkcji nieczystych 45 ■ Dlaczego testowanie nieczystych funkcji jest trudne 46 ■ Parametryzowane testy jednostkowe 48 ■ Unikanie interfejsów nagłówkowych 49

2.4 Czystość a ewolucja obliczeń 52

3 Projektowanie sygnatur funkcji i typów 54

3.1 Projektowanie sygnatury funkcji 55

Notacja strzałkowa 55 ■ Ile informacji niesie sygnatura? 56

3.2 Przechwytywanie danych za pomocą obiektów danych 57

Typy podstawowe często nie są dostatecznie konkretne 58 ■ Ograniczenie wejść za pomocą typów niestandardowych 59 ■ Pisanie „uczciwych” funkcji 61 ■ Łączenie wartości za pomocą krotek i obiektów 62

3.3 Modelowanie braku danych za pomocą Unit 63

Dlaczego void nie jest idealny 64 ■ Usuwanie luki między Action a Func za pomocą Unit 65

3.4 Modelowanie możliwego braku danych za pomocą Option 67

Kiepskie API, których używamy na co dzień 67 ■ Wprowadzenie do typu Option 68 ■ Implementacja Option 71 ■ Większa niezawodność dzięki zastosowaniu Option zamiast null 74 ■ Option jako naturalny typ wyniku funkcji częściowych 75

4 Wzorce w programowaniu funkcyjnym 82

4.1 Stosowanie funkcji do wewnętrznych wartości struktury 82

Odwzorowanie funkcji na elementach ciągu 83 ■ Odwzorowanie funkcji na Option 83 ■ Jak Option zwiększa poziom abstrakcji 86 ■ Wprowadzenie do funk-torów 87

4.2	Wykonywanie efektów ubocznych za pomocą ForEach	88
4.3	Łączenie funkcji za pomocą Bind	90
	Połączenie ze sobą funkcji zwracających Option	91
	Spłaszczanie zagnieżdżonych list za pomocą Bind	92
	To monada!	93
	Funkcja Return	94
	Relacje między funktorami a monadami	95
4.4	Filtrowanie wartości za pomocą Where	95
4.5	Połączenie Option i IEnumerable za pomocą Bind	96
4.6	Kodowanie na różnych poziomach abstrakcji	98
	Wartości regularne a podniesione	99
	Przekraczanie poziomów abstrakcji	100
	Map kontra Bind – ponownie	101
	Praca na właściwym poziomie abstrakcji	102
5	Projektowanie programów za pomocą składania funkcji	104
5.1	Złożenie funkcji	105
	Przegląd informacji o złożeniu funkcji	105
	Łłańcuch metod	106
	Złożenie w świecie podwyższonego poziomu	106
5.2	Myślenie w kategoriach przepływu danych	107
	Używanie składalnego API z LINQ	108
	Pisanie funkcji, które łatwo podlegają złożeniu	109
5.3	Programowanie przepływów pracy	111
	Prosty przepływ pracy w celu weryfikacji	112
	Refaktoryzacja z uwzględnieniem przepływu danych	113
	Złożenie prowadzi do większej elastyczności	114
5.4	Wprowadzenie do funkcyjnego modelowania dziedziny	115
5.5	Kompleksowy przepływ pracy po stronie serwera	116
	Wyrażenia kontra instrukcje	118
	Deklaratywnie kontra imperatywnie	119
	Funkcyjne spojrzenie na warstwy	120
CZĘŚĆ II W STRONĘ FUNKCYJNOŚCI		123
6	Funkcyjne obsługiwane błędów	125
6.1	Bezpieczniejszy sposób przedstawiania wyników	126
	Uchwycenie szczegółów dotyczących błędów za pomocą Either	127
	Podstawowe funkcje do wykorzystania z Either	130
	Porównanie Option i Either	131
6.2	Łączenie działań, które mogą się nie udać	132
6.3	Walidacja: doskonały przypadek zastosowania Either	135
	Wybór odpowiedniej reprezentacji dla błędów	135
	Definiowanie API opartego na Either	136
	Dodawanie kodu walidacji	137

- 6.4 Przedstawianie wyników aplikacjom klienckim 138
 - Udostępnianie interfejsu typu Option 140 ■ Udostępnianie interfejsu typu Either 141 ■ Zwracanie wynikowego obiektu (DTO) 142
- 6.5 Wariacje na temat Either 143
 - Zmiana reprezentacji błędów 144 ■ Specjalizowane wersje Either 145 ■ Refaktoryzacja do Validation i Exceptional 146 ■ Odejście od wyjątków? 149

7 Budowanie aplikacji za pomocą funkcji 152

- 7.1 Aplikacja częściowa: dostarczanie fragmentów argumentów 153
 - Ręczne włączanie częściowej aplikacji 155 ■ Uogólnianie aplikacji częściowej 156 ■ Kolejność argumentów ma znaczenie 157
- 7.2 Pokonywanie kaprysów rozwiązywania metod 158
- 7.3 Funkcje rozwinięte: zoptymalizowane pod kątem częściowej aplikacji 160
- 7.4 Tworzenie API przyjaznego dla aplikacji częściowej 163
 - Typy jako dokumentacja 165 ■ Wyszczególnianie funkcji dostępu do danych 166
- 7.5 Modularyzacja i budowanie aplikacji 168
 - Modułowość w programowaniu obiektowym 168 ■ Modułowość w FP 170 ■ Porównanie dwóch podejść 173 ■ Budowanie aplikacji 174
- 7.6 Redukowanie listy do jednej wartości 175
 - Metoda Aggregate w LINQ 175 ■ Agregowanie wyników walidacji 177 ■ Zbieranie błędów walidacji 178

8 Efektywne wykorzystywanie funkcji wieloargumentowych 181

- 8.1 Aplikacja funkcji w świecie podniesionych typów 182
 - Zrozumienie aplikatyw 184 ■ Podnoszenie poziomu funkcji 186 ■ Wprowadzenie do testowania opartego na własnościach 187
- 8.2 Funktory, aplikatywy, monady 190
- 8.3 Prawa monad 192
 - Prawa tożsamość 192 ■ Lewa tożsamość 192 ■ Łączność 193 ■ Używanie Bind z funkcjami wieloargumentowymi 195
- 8.4 Poprawianie czytelności przez użycie LINQ z dowolną monadą 195
 - Korzystanie z LINQ z arbitralnymi funktorami 196 ■ Używanie LINQ dowolnymi monadami 197 ■ let, where i inne klauzule LINQ 201

8.5 Kiedy używać Bind, a kiedy Apply 202

Walidacja za pomocą inteligentnych konstruktorów 202 ■ Zbieranie błędów za pomocą przepływu aplikatywnego 203 ■ Szybka porażka przy przepływie monadycznym 205

9 Rozważania o funkcyjności danych 208

9.1 Pułapki mutacji stanu 209

9.2 Zrozumienie stanu, tożsamości i zmiany 212

Niektóre rzeczy nigdy się nie zmieniają 212 ■ Reprezentowanie zmian bez mutacji 214

9.3 Wymuszanie niemutowalności 217

Całkowita niemutowalność 219 ■ Metody kopiowania bez szablonowego kodu? 220 ■ Wykorzystywanie F# do typów danych 222 ■ Porównywanie strategii dla niemutowalności: konkurs brzydoty 224

9.4 Krótkie wprowadzenie do funkcyjnych struktur danych 225

Klasyczna lista wiązana w stylu funkcyjnym 226 ■ Drzewa binarne 231

10 Event sourcing: funkcyjne podejście do zapisu 237

10.1 Funkcyjne rozważania o przechowywaniu danych 238

Dlaczego przechowywanie danych powinno polegać wyłącznie na ich dołączaniu 239 ■ Zrelaksujmy się i zapomnijmy o przechowywaniu stanu 239

10.2 Podstawy event sourcingu 241

Reprezentacja zdarzeń 241 ■ Przechowywanie zdarzeń 242 ■ Reprezentacja stanu 243 ■ Przerwa na dopasowywanie do wzorca 244 ■ Reprezentacja zmian stanu 247 ■ Odtwarzanie bieżącego stanu na podstawie przeszłych zdarzeń 248

10.3 Architektura systemu opartego na zdarzeniach 249

Obsługa poleceń 251 ■ Obsługa zdarzeń 254 ■ Dodanie walidacji 255 ■ Tworzenie widoków danych na podstawie zdarzeń 257

10.4 Porównanie różnych podejść do niemutowalnego magazynu 260

Datomic kontra Event Store 261 ■ Jak bardzo twoja dziedzina jest sterowana zdarzeniami? 262

CZĘŚĆ III ZAAWANSOWANE TECHNIKI 265

11 Leniwe obliczenia, kontynuacje oraz piękno kompozycji monadycznej 267

11.1 Zalety leniwego wartościowania 268

Leniwe API działające z Option 268 ■ Złożenie leniwych wartościowań 271

11.2 Obsługa wyjątków za pomocą Try 273

Reprezentowanie obliczeń, które mogą się nie powieść 273 ■ Bezpieczne pobieranie informacji z obiektu JSON 274 ■ Złożenia obliczeń, które mogą się nie udać 275 ■ Złożenie monadyczne: co to znaczy? 277

11.3 Tworzenie potoku oprogramowania pośredniczącego na potrzeby dostępu do bazy danych 278

Złożenie funkcji, które inicjalizują/czyszczą 278 ■ Przepis na uniknięcie piramidy potępienia 280 ■ Przechwycenie istoty oprogramowania pośredniczącego 280 ■ Implementacja wzorca zapytania dla oprogramowania pośredniczącego 282 ■ Dodawanie oprogramowania pośredniczącego, które mierzy czas działania 286 ■ Dodawanie oprogramowania pośredniczącego zarządzającego transakcją 287

12 Stanowe programy i obliczenia 290

12.1 Programy, które zarządzają stanem 291

Utrzymywanie pamięci podręcznej uzyskanych zasobów 292 ■ Refaktoryzacja w celu umożliwienia testowania i obsługi błędów 294 ■ Obliczenia stanowe 296

12.2 Język do generowania danych losowych 297

Generowanie całkowitych liczb losowych 298 ■ Generowanie innych wartości podstawowych 298 ■ Generowanie bardziej złożonych struktur 300

12.3 Ogólny wzorzec obliczeń stanowych 302

13 Posługiwanie się obliczeniami asynchronicznymi 306

13.1 Obliczenia asynchroniczne 307

Potrzeba działania asynchronicznego 307 ■ Reprezentowanie działań asynchronicznych za pomocą Task 308 ■ Zadanie jako kontener na przyszłą wartość 309 ■ Obsługa błędów 311 ■ API HTTP API dla konwersji walut 313 ■ W razie niepowodzenia próbujmy kilka razy 314 ■ Równoległe uruchamianie działań asynchronicznych 315

13.2 Funkcje przesuwne: praca z listami podniesionych wartości 317

Weryfikacja listy wartości za pomocą monadycznej funkcji Traverse 319 ■ Zbieranie błędów walidacji za pomocą aplikatywnej Traverse 320 ■ Zastosowanie wielu walidatorów do jednej wartości 322 ■ Stosowanie Traverse z Task, jeśli możliwych jest wiele wyników 323 ■ Definiowanie Traverse dla struktur o pojedynczej wartości 325

13.3 Łączenie asynchroniczności i walidacji (lub dowolnych innych efektów monadycznych) 326

Problem złożenia monad 326 ■ Zmniejszenie liczby efektów 328 ■ Wyrażenia LINQ ze złożeniem monad 329

- 14 Strumienie danych i Reactive Extensions 332**
 - 14.1 Reprezentowanie strumieni danych za pomocą IObservable 333
 - Ciąg wartości w czasie 333 ■ Subskrypcja IObservable 335
 - 14.2 Tworzenie IObservables 336
 - Tworzenie zegara 336 ■ Używanie Subject, aby poinformować IObservable, kiedy ma wysłać sygnał 337 ■ Tworzenie IObservables na podstawie subskrypcji opartych na wywołaniach zwrotnych 338 ■ Tworzenie IObservables z prostszych struktur 339
 - 14.3 Przekształcanie i łączenie strumieni danych 340
 - Przekształcenia strumienia 340 ■ Łączenie i podział strumieni 342 ■ Obsługa błędów w IObservable 344 ■ Składamy wszystko razem 346
 - 14.4 Implementacja kodu, który łączy wiele zdarzeń 347
 - Wykrywanie ciągów naciśnięć klawiszy 348 ■ Reagowanie na wiele źródeł zdarzeń 350 ■ Powiadamianie, kiedy na koncie pojawia się debet 352
 - 14.5 Kiedy należy stosować IObservable? 355
- 15 Wprowadzenie do współbieżności z przesyłaniem komunikatów 357**
 - 15.1 Zapotrzebowanie na współdzielony stan mutowalny 358
 - 15.2 Zrozumienie współbieżności z przesyłaniem komunikatów 359
 - Implementowanie agentów w C# 362 ■ Pierwsze kroki z agentami 363 ■ Użycie agentów do obsługi równoczesnych żądań 365 ■ Agenty kontra aktry 368
 - 15.3 Funkcyjne API, implementacje oparte na agentach 370
 - Agenty jako szczegóły implementacji 371 ■ Ukrywanie agentów za tradycyjnym API 372
 - 15.4 Współbieżność z przesyłaniem komunikatów w aplikacjach LOB 373
 - Korzystanie z agenta do synchronizacji dostępu do danych konta 374 ■ Przechowywanie rejestru kont 376 ■ Agent nie jest obiektem 377 ■ Łączenie wszystkiego ze sobą 380
- Epilog: co dalej? 383**