

Przedmowa (13)

Podziękowania (17)

O książce (19)

Rozdział 1. Czym jest programowanie funkcyjne? (23)

- 1.1. Czym jest programowanie funkcyjne? (24)
- 1.2. Pisanie użytecznych programów bez efektów ubocznych (26)
- 1.3. W jaki sposób transparentność referencyjna czyni program bezpieczniejszym? (28)
- 1.4. Zalety programowania funkcyjnego (28)
- 1.5. Wykorzystanie modelu z zastępowaniem do rozumowania na temat programu (30)
- 1.6. Zastosowanie zasad funkcyjnych na prostym przykładzie (31)
- 1.7. Osiąganie limitów abstrakcji (36)
- 1.8. Podsumowanie (37)

Rozdział 2. Użycie funkcji w języku Java (39)

- 2.1. Czym jest funkcja? (40)
 - 2.1.1. Funkcje w świecie rzeczywistym (40)
- 2.2. Funkcje w Javie (45)
 - 2.2.1. Metody funkcyjne (45)
 - 2.2.2. Interfejsy funkcyjne Javy i klasy anonimowe (50)
 - 2.2.3. Złożenie funkcji (52)
 - 2.2.4. Funkcje polimorficzne (52)
 - 2.2.5. Upraszczanie kodu za pomocą funkcji anonimowych (53)
- 2.3. Zaawansowane funkcjonalności funkcji (55)
 - 2.3.1. Co z funkcjami dotyczącymi kilku argumentów? (56)
 - 2.3.2. Zastosowanie funkcji z częściowym rozwinięciem (57)
 - 2.3.3. Funkcje wyższego rzędu (57)
 - 2.3.4. Polimorficzne funkcje wyższego rzędu (58)
 - 2.3.5. Użycie funkcji anonimowych (61)
 - 2.3.6. Funkcje lokalne (63)
 - 2.3.7. Domknięcia (64)
 - 2.3.8. Częściowe zastosowanie funkcji i automatyczne rozwijanie (66)
 - 2.3.9. Zamiana argumentów częściowo zastosowanych funkcji (70)
 - 2.3.10. Funkcje rekurencyjne (71)
 - 2.3.11. Funkcja tożsamościowa (73)
- 2.4. Interfejsy funkcyjne Javy 8 (74)
- 2.5. Debugging funkcji anonimowych (75)
- 2.6. Podsumowanie (78)

Rozdział 3. Uczynić Javę bardziej funkcyjną (79)

- 3.1. Zamiana standardowych struktur sterujących na ich funkcyjne odpowiedniki (80)
- 3.2. Abstrakcja struktur sterujących (81)
 - 3.2.1. Czyszczenie kodu (85)
 - 3.2.2. Alternatywa dla if ... else (88)
- 3.3. Abstrakcja iteracji (92)
 - 3.3.1. Abstrakcja operacji na liście dzięki odwzorowaniu (94)
 - 3.3.2. Tworzenie list (95)
 - 3.3.3. Wykorzystanie operacji dotyczących głowy i ogona (96)

- 3.3.4. Funkcyjne dodawanie do listy (97)
- 3.3.5. Redukcja i zwijanie list (97)
- 3.3.6. Kompozycja odwzorowań i mapowanie kompozycji (103)
- 3.3.7. Stosowanie efektów dla list (104)
- 3.3.8. Funkcyjne podejście do danych wyjściowych (105)
- 3.3.9. Budowanie list referencji odwrotnych (106)
- 3.4. Zastosowanie właściwych typów (109)
 - 3.4.1. Problemy ze standardowymi typami (109)
 - 3.4.2. Definiowanie typów wartości (112)
 - 3.4.3. Przyszłość typów wartości w Javie (115)
- 3.5. Podsumowanie (115)

Rozdział 4. Rekurencja, rekurencja odwrotna i memoizacja (117)

- 4.1. Różnice między rekurencją i rekurencją odwrotną (118)
 - 4.1.1. Przykład z dodawaniem dla obu rodzajów rekurencji (118)
 - 4.1.2. Implementacja rekurencji w Javie (119)
 - 4.1.3. Wykorzystanie eliminacji wywołania ogonowego (119)
 - 4.1.4. Użycie funkcji i metod z rekurencją ogonową (120)
 - 4.1.5. Abstrakcja rekurencji (120)
 - 4.1.6. Utworzenie wersji zapewniającej prostą podmiannę metody rekurencyjnej bazującej na stosie (124)
- 4.2. Stosowanie funkcji rekurencyjnych (126)
 - 4.2.1. Korzystanie z lokalnie zdefiniowanych funkcji (127)
 - 4.2.2. Zapewnienie funkcji działających jako rekurencje ogonowe (128)
 - 4.2.3. Funkcje podwójnie rekurencyjne - ciąg Fibonacciego (128)
 - 4.2.4. Zamiana metod dla list na wersje rekurencyjne i bezpieczne dla stosu (131)
- 4.3. Kompozycja ogromnej liczby funkcji (134)
- 4.4. Korzystanie z memoizacji (137)
 - 4.4.1. Memoizacja w programowaniu imperatywnym (137)
 - 4.4.2. Memoizacja w funkcjach rekurencyjnych (138)
 - 4.4.3. Memoizacja automatyczna (140)
- 4.5. Podsumowanie (146)

Rozdział 5. Obsługa danych przy użyciu list (147)

- 5.1. Jak klasyfikować kolekcje danych? (147)
 - 5.1.1. Różne rodzaje list (148)
 - 5.1.2. Względna oczekiwana wydajność listy (149)
 - 5.1.3. Wymiana czasu na zajętość pamięci lub czasu kontra złożoność (150)
 - 5.1.4. Modyfikacja na miejscu (151)
 - 5.1.5. Trwałe struktury danych (152)
- 5.2. Implementacja niezmienniej, trwałej listy jednokierunkowej (153)
- 5.3. Współdzielenie danych w operacjach na liście (156)
 - 5.3.1. Dodatkowe operacje na liście (158)
- 5.4. Wykorzystanie rekurencji do zwijania list za pomocą funkcji wyższego rzędu (163)
 - 5.4.1. Bazująca na sterce, rekurencyjna wersja foldRight (169)
 - 5.4.2. Odwzorowanie i filtrowanie list (171)
- 5.5. Podsumowanie (173)

Rozdział 6. Obsługa danych opcjonalnych (175)

- 6.1. Problemy ze wskaźnikiem null (176)
- 6.2. Alternatywy dla referencji null (177)
- 6.3. Typ danych Option (180)

- 6.3.1. Pobranie wartości z Option (182)
- 6.3.2. Stosowanie funkcji dla wartości opcjonalnych (184)
- 6.3.3. Kompozycja obiektów Option (185)
- 6.3.4. Sposoby użycia Option (187)
- 6.3.5. Inne sposoby łączenia opcji (191)
- 6.3.6. Kompozycja List z Option (193)
- 6.4. Różne narzędzia dodatkowe dla Option (195)
 - 6.4.1. Testowanie, czy to Some, czy None (195)
 - 6.4.2. Implementacja metod equals i hashCode (195)
- 6.5. Jak i gdzie używać Option? (196)
- 6.6. Podsumowanie (199)

Rozdział 7. Obsługa błędów i wyjątków (201)

- 7.1. Problemy do rozwiązania (201)
- 7.2. Typ Either (203)
 - 7.2.1. Kompozycja klasy Either (204)
- 7.3. Typ Result (206)
 - 7.3.1. Dodawanie metod do klasy Result (207)
- 7.4. Wzorce Result (209)
- 7.5. Zaawansowana obsługa Result (216)
 - 7.5.1. Stosowanie predykatów (216)
 - 7.5.2. Mapowanie porażek (217)
 - 7.5.3. Dodanie metod fabrycznych (220)
 - 7.5.4. Stosowanie efektów (221)
 - 7.5.5. Zaawansowana kompozycja wyników (224)
- 7.6. Podsumowanie (227)

Rozdział 8. Zaawansowana obsługa list (229)

- 8.1. Problem z length (230)
 - 8.1.1. Problem wydajności (230)
 - 8.1.2. Zalety memoizacji (231)
 - 8.1.3. Wady memoizacji (231)
 - 8.1.4. Faktyczna wydajność (233)
- 8.2. Kompozycja List i Result (233)
 - 8.2.1. Metody List zwracające Result (233)
 - 8.2.2. Konwersja z List<Result> na Result<List> (235)
- 8.3. Abstrakcja typowych operacji na listach (238)
 - 8.3.1. Zszywanie i rozszywanie list (238)
 - 8.3.2. Dostęp do elementów na podstawie ich indeksów (241)
 - 8.3.3. Dzielenie list (243)
 - 8.3.4. Poszukiwanie podlist (247)
 - 8.3.5. Różnorakie funkcje dotyczące obsługi list (248)
- 8.4. Automatyczne przetwarzanie równoległe list (251)
 - 8.4.1. Nie wszystkie obliczenia można zrównoleglić (251)
 - 8.4.2. Podział listy na podlisty (252)
 - 8.4.3. Zrównoleglone przetwarzanie podlist (253)
- 8.5. Podsumowanie (255)

Rozdział 9. Wykorzystywanie leniwości obliczeń (257)

- 9.1. Zrozumieć rygor i lenistwo (258)
 - 9.1.1. Java jest językiem rygorystycznym (258)
 - 9.1.2. Problem z rygorem (259)

- 9.2. Implementacja wersji leniwej (261)
- 9.3. Rzeczy, których nie wykonamy bez lenistwa (262)
- 9.4. Dlaczego nie użyjemy klasy Stream z Javy 8? (263)
- 9.5. Tworzenie struktury danych dla leniwej listy (263)
 - 9.5.1. Memoizacja wyliczonych wartości (265)
 - 9.5.2. Modyfikacja strumienia (268)
- 9.6. Prawdziwa esencja lenistwa (271)
 - 9.6.1. Zwijanie strumieni (273)
- 9.7. Obsługa strumieni nieskończonych (278)
- 9.8. Unikanie referencji null i modyfikowalnych pól (280)
- 9.9. Podsumowanie (282)

Rozdział 10. Obsługa danych za pomocą drzew (285)

- 10.1. Drzewo binarne (286)
 - 10.1.1. Drzewa zrównoważone i niezbalansowane (287)
 - 10.1.2. Rozmiar, wysokość i głębokość (287)
 - 10.1.3. Drzewa liściaste (288)
 - 10.1.4. Uporządkowane drzewa binarne lub też drzewa binarne wyszukiwania (288)
 - 10.1.5. Kolejność wstawiania (289)
 - 10.1.6. Kolejność przejścia przez drzewo (290)
- 10.2. Implementacja drzewa binarnego (292)
- 10.3. Usuwanie elementów z drzew (298)
- 10.4. Łączenie dowolnych drzew (300)
- 10.5. Zwijanie drzewa (304)
 - 10.5.1. Zwijanie za pomocą dwóch funkcji (305)
 - 10.5.2. Zwijanie za pomocą jednej funkcji (307)
 - 10.5.3. Którą implementację zwinięcia wybrać? (308)
- 10.6. Odwzorowanie drzew (310)
- 10.7. Równoważenie drzew (311)
 - 10.7.1. Obracanie drzew (311)
 - 10.7.2. Równoważenie drzew za pomocą algorytmu Day-Stout-Warren (314)
 - 10.7.3. Automatycznie równoważące się drzewa (315)
 - 10.7.4. Rozwiązywanie właściwego problemu (316)
- 10.8. Podsumowanie (317)

Rozdział 11. Rozwiązywanie rzeczywistych problemów przy użyciu zaawansowanych drzew (319)

- 11.1. Lepsza wydajność i bezpieczeństwo stosu dzięki samobalansującym się drzewom (320)
 - 11.1.1. Prosta struktura drzewa (320)
 - 11.1.2. Wstawianie elementu do drzewa czerwono-czarnego (325)
- 11.2. Przykład użycia drzew czerwono-czarnych - mapowanie (330)
 - 11.2.1. Implementacja klasy Map (330)
 - 11.2.2. Rozbudowania klasy Map (333)
 - 11.2.3. Użycie klasy Map dla kluczy bez możliwości porównywania (334)
- 11.3. Implementacja funkcyjnej kolejki priorytetowej (336)
 - 11.3.1. Protokół dostępu dla kolejki priorytetowej (336)
 - 11.3.2. Sposoby użycia kolejek priorytetowych (337)
 - 11.3.3. Wymagania implementacyjne (337)
 - 11.3.4. Struktura danych nazywana kopcem lewostronnym (338)
 - 11.3.5. Implementacja kopca lewostronnego (338)
 - 11.3.6. Implementacja interfejsu przypominającego kolejkę (343)
- 11.4. Kolejka priorytetowa dla elementów bez możliwości porównywania (344)
- 11.5. Podsumowanie (349)

Rozdział 12. Obsługa zmian stanu w sposób funkcyjny (351)

- 12.1. Funkcjonalny generator liczb losowych (352)
 - 12.1.1. Interfejs generatora liczb losowych (353)
 - 12.1.2. Implementacja generatora liczb losowych (354)
- 12.2. Ogólne API do obsługi stanu (357)
 - 12.2.1. Korzystanie z operacji na stanie (358)
 - 12.2.2. Kompozycja operacji na stanie (359)
- 12.3. Ogólna obsługa stanu (363)
 - 12.3.1. Wzorce stanu (364)
 - 12.3.2. Tworzenie maszyny stanowej (365)
 - 12.3.3. Kiedy korzystać ze stanu i maszyny stanowej (370)
- 12.4. Podsumowanie (371)

Rozdział 13. Funkcyjne wejście-wyjście (373)

- 13.1. Stosowanie efektów w kontekście (374)
 - 13.1.1. Czym są efekty? (374)
 - 13.1.2. Implementacja efektów (375)
 - 13.1.3. Bardziej użyteczne efekty dla porażek (377)
- 13.2. Odczyt danych (380)
 - 13.2.1. Odczyt danych z konsoli (380)
 - 13.2.2. Odczyt danych z pliku (384)
 - 13.2.3. Testowanie z zadanymi danymi wejściowymi (386)
- 13.3. Naprawdę funkcyjne wejście-wyjście (387)
 - 13.3.1. W jaki sposób zapewnić pełną funkcyjność wejścia-wyjścia? (387)
 - 13.3.2. Implementacja w pełni funkcyjnego wejścia-wyjścia (388)
 - 13.3.3. Łączenie operacji wejścia-wyjścia (389)
 - 13.3.4. Obsługa wejścia za pomocą IO (390)
 - 13.3.5. Rozszerzanie typu IO (393)
 - 13.3.6. Uczynienie typu IO bezpiecznym dla stosu (395)
- 13.4. Podsumowanie (400)

Rozdział 14. Współdzielenie zmiennego stanu przy użyciu aktorów (401)

- 14.1. Model aktora (402)
 - 14.1.1. Asynchroniczne komunikaty (403)
 - 14.1.2. Obsługa zrównoleglenia (403)
 - 14.1.3. Obsługa zmiany stanu aktora (404)
- 14.2. Budowanie frameworka aktora (405)
 - 14.2.1. Ograniczenia prezentowanego frameworka aktora (405)
 - 14.2.2. Projektowanie interfejsów frameworka aktorów (405)
 - 14.2.3. Implementacja AbstractActor (407)
- 14.3. Zmuszenie aktorów do działania (408)
 - 14.3.1. Implementacja przykładu z ping-pongiem (409)
 - 14.3.2. Bardziej poważny przykład - równoległe wykonywanie obliczeń (410)
 - 14.3.3. Zmiana kolejności wyników (415)
 - 14.3.4. Rozwiązanie problemu wydajności (418)
- 14.4. Podsumowanie (423)

Rozdział 15. Rozwiązywanie typowych problemów w sposób funkcyjny (425)

- 15.1. Wykorzystanie asercji do walidacji danych (426)
- 15.2. Odczyt właściwości z pliku (430)
 - 15.2.1. Wczytywanie pliku właściwości (430)

- 15.2.2. Odczyt właściwości jako tekstu (431)
- 15.2.3. Tworzenie lepszych komunikatów o błędzie (432)
- 15.2.4. Odczyt właściwości jako listy (435)
- 15.2.5. Odczytywanie wyliczeń (436)
- 15.2.6. Odczyt właściwości dowolnych typów (437)
- 15.3. Konwersja programu imperatywnego - czytnik plików XML (440)
 - 15.3.1. Zebranie potrzebnych funkcji (441)
 - 15.3.2. Kompozycja funkcji i stosowanie efektu (442)
 - 15.3.3. Implementacja funkcji (443)
 - 15.3.4. Uczyńnienie programu nawet bardziej funkcyjnym (444)
 - 15.3.5. Rozwiązanie problemu z typem argumentu (448)
 - 15.3.6. Zmiana funkcji przetwarzającej element na parametr (449)
 - 15.3.7. Obsługa błędów dla nazw elementów (450)
- 15.4. Podsumowanie (451)

Dodatek A. Wykorzystanie elementów funkcyjnych Javy 8 (453)

- A.1. Klasa Optional (454)
- A.2. Strumienie (455)

Dodatek B. Monady (461)

Dodatek C. Co dalej? (467)

- C.1. Wybór nowego języka (467)
 - C.1.1. Haskell (467)
 - C.1.2. Scala (468)
 - C.1.3. Kotlin (468)
 - C.1.4. Frege (469)
 - C.1.5. A co z dynamicznie typowanymi językami funkcyjnymi? (469)
- C.2. Pozostanie z Javą (469)
 - C.2.1. Functional Java (470)
 - C.2.2. Javaslang (470)
 - C.2.3. Cyclops (470)
 - C.2.4. Inne biblioteki funkcyjne (471)
- C.3. Dodatkowe lektury (471)

Skorowidz (473)