Contents

Contents **xiii**

List of Figures **xxiii**

List of Tables **xxv**

List of Listings **xxvii**

Foreword **xxxv**

Acknowledgments **xxxvii**

Introduction **xli**

1 A Scalable Language **3**

1.1 A language that grows on you 4

1.2 What makes Scala scalable? 9

1.3 Why Scala? 12

1.4 Scala's roots 19

1.5 Conclusion 21

2 First Steps in Scala **23**

Step 1. Learn to use the Scala interpreter . . . 23

Step 2. Define some variables 25

Step 3. Define some functions 27

Step 4. Write some Scala scripts 29

Step 5. Loop with `while`; decide with `if` 30

Step 6. Iterate with `foreach` and `for` 32

Conclusion 35

3 Next Steps in Scala **37**
Step 7. Parameterize arrays with types 37
Step 8. Use lists 41
Step 9. Use tuples 46
Step 10. Use sets and maps 47
Step 11. Learn to recognize the functional style 52
Step 12. Read lines from a file 55
Conclusion 58

4 Classes and Objects **59**
4.1 Classes, fields, and methods 59
4.2 Semicolon inference 64
4.3 Singleton objects 65
4.4 A Scala application 68
4.5 The `App` trait 70
4.6 Conclusion 71

5 Basic Types and Operations **73**
5.1 Some basic types 73
5.2 Literals 74
5.3 String interpolation 80
5.4 Operators are methods 82
5.5 Arithmetic operations 85
5.6 Relational and logical operations 86
5.7 Bitwise operations 88
5.8 Object equality 89
5.9 Operator precedence and associativity 91
5.10 Rich wrappers 94
5.11 Conclusion 94

6 Functional Objects **97**
6.1 A specification for class `Rational` 97
6.2 Constructing a `Rational` 98
6.3 Reimplementing the `toString` method 100
6.4 Checking preconditions 101
6.5 Adding fields 101
6.6 Self references 103
6.7 Auxiliary constructors 104

6.8 Private fields and methods 106
6.9 Defining operators 107
6.10 Identifiers in Scala 109
6.11 Method overloading 112
6.12 Implicit conversions 114
6.13 A word of caution 115
6.14 Conclusion 115

7 Built-in Control Structures **117**
7.1 If expressions 118
7.2 While loops 119
7.3 For expressions 122
7.4 Exception handling with `try` expressions 127
7.5 Match expressions 131
7.6 Living without `break` and `continue` 133
7.7 Variable scope 135
7.8 Refactoring imperative-style code 139
7.9 Conclusion 141

8 Functions and Closures **143**
8.1 Methods 143
8.2 Local functions 145
8.3 First-class functions 147
8.4 Short forms of function literals 149
8.5 Placeholder syntax 150
8.6 Partially applied functions 151
8.7 Closures 154
8.8 Special function call forms 158
8.9 Tail recursion 161
8.10 Conclusion 165

9 Control Abstraction **167**
9.1 Reducing code duplication 167
9.2 Simplifying client code 171
9.3 Currying 173
9.4 Writing new control structures 175
9.5 By-name parameters 178
9.6 Conclusion 181

10 Composition and Inheritance **183**
10.1 A two-dimensional layout library 183
10.2 Abstract classes 184
10.3 Defining parameterless methods 185
10.4 Extending classes 188
10.5 Overriding methods and fields 190
10.6 Defining parametric fields 191
10.7 Invoking superclass constructors 193
10.8 Using override modifiers 194
10.9 Polymorphism and dynamic binding 196
10.10 Declaring final members 198
10.11 Using composition and inheritance 200
10.12 Implementing above, beside, and toString 201
10.13 Defining a factory object 203
10.14 Heighten and widen 205
10.15 Putting it all together 209
10.16 Conclusion 210

11 Scala's Hierarchy **211**
11.1 Scala's class hierarchy 211
11.2 How primitives are implemented 215
11.3 Bottom types 217
11.4 Defining your own value classes 218
11.5 Conclusion 221

12 Traits **223**
12.1 How traits work 223
12.2 Thin versus rich interfaces 226
12.3 Example: Rectangular objects 227
12.4 The Ordered trait 230
12.5 Traits as stackable modifications 232
12.6 Why not multiple inheritance? 236
12.7 To trait or not to trait? 240
12.8 Conclusion 241

13 Packages and Imports **243**
13.1 Putting code in packages 243
13.2 Concise access to related code 244

13.3 Imports 248
13.4 Implicit imports 252
13.5 Access modifiers 253
13.6 Package objects 258
13.7 Conclusion 260

14 Assertions and Tests **261**
14.1 Assertions 261
14.2 Testing in Scala 263
14.3 Informative failure reports 264
14.4 Tests as specifications 266
14.5 Property-based testing 269
14.6 Organizing and running tests 271
14.7 Conclusion 272

15 Case Classes and Pattern Matching **273**
15.1 A simple example 273
15.2 Kinds of patterns 278
15.3 Pattern guards 287
15.4 Pattern overlaps 288
15.5 Sealed classes 290
15.6 The `Option` type 292
15.7 Patterns everywhere 294
15.8 A larger example 298
15.9 Conclusion 306

16 Working with Lists **307**
16.1 List literals 307
16.2 The `List` type 308
16.3 Constructing lists 308
16.4 Basic operations on lists 309
16.5 List patterns 310
16.6 First-order methods on class `List` 312
16.7 Higher-order methods on class `List` 324
16.8 Methods of the `List` object 332
16.9 Processing multiple lists together 334
16.10 Understanding Scala's type inference algorithm . . . 335
16.11 Conclusion 339

17 Working with Other Collections **341**
17.1 Sequences 341
17.2 Sets and maps 345
17.3 Selecting mutable versus immutable collections . . . 353
17.4 Initializing collections 356
17.5 Tuples 360
17.6 Conclusion 362

18 Mutable Objects **363**
18.1 What makes an object mutable? 363
18.2 Reassignable variables and properties 366
18.3 Case study: Discrete event simulation 369
18.4 A language for digital circuits 370
18.5 The `Simulation` API 373
18.6 Circuit Simulation 377
18.7 Conclusion 385

19 Type Parameterization **387**
19.1 Functional queues 387
19.2 Information hiding 391
19.3 Variance annotations 394
19.4 Checking variance annotations 398
19.5 Lower bounds 401
19.6 Contravariance 403
19.7 Object private data 406
19.8 Upper bounds 408
19.9 Conclusion 411

20 Abstract Members **413**
20.1 A quick tour of abstract members 413
20.2 Type members 414
20.3 Abstract `val`s 415
20.4 Abstract `var`s 416
20.5 Initializing abstract `val`s 417
20.6 Abstract `type`s 425
20.7 Path-dependent types 427
20.8 Refinement types 430
20.9 Enumerations 431

17 Working with Other Collections **341**
17.1 Sequences . . . 341
17.2 Sets and maps . . . 345
17.3 Selecting mutable versus immutable collections . . . 353
17.4 Initializing collections . . . 356
17.5 Tuples . . . 360
17.6 Conclusion . . . 362

18 Mutable Objects **363**
18.1 What makes an object mutable? . . . 363
18.2 Reassignable variables and properties . . . 366
18.3 Case study: Discrete event simulation . . . 369
18.4 A language for digital circuits . . . 370
18.5 The `Simulation` API . . . 373
18.6 Circuit Simulation . . . 377
18.7 Conclusion . . . 385

19 Type Parameterization **387**
19.1 Functional queues . . . 387
19.2 Information hiding . . . 391
19.3 Variance annotations . . . 394
19.4 Checking variance annotations . . . 398
19.5 Lower bounds . . . 401
19.6 Contravariance . . . 403
19.7 Object private data . . . 406
19.8 Upper bounds . . . 408
19.9 Conclusion . . . 411

20 Abstract Members **413**
20.1 A quick tour of abstract members . . . 413
20.2 Type members . . . 414
20.3 Abstract `val`s . . . 415
20.4 Abstract `var`s . . . 416
20.5 Initializing abstract `val`s . . . 417
20.6 Abstract `type`s . . . 425
20.7 Path-dependent types . . . 427
20.8 Refinement types . . . 430
20.9 Enumerations . . . 431

20.10 Case study: Currencies 433
20.11 Conclusion 443

21 Implicit Conversions and Parameters 445
21.1 Implicit conversions 445
21.2 Rules for implicits 448
21.3 Implicit conversion to an expected type 451
21.4 Converting the receiver 453
21.5 Implicit parameters 456
21.6 Context bounds 462
21.7 When multiple conversions apply 465
21.8 Debugging implicits 467
21.9 Conclusion 468

22 Implementing Lists 471
22.1 The `List` class in principle 471
22.2 The `ListBuffer` class 477
22.3 The `List` class in practice 479
22.4 Functional on the outside 481
22.5 Conclusion 482

23 For Expressions Revisited 485
23.1 For expressions 486
23.2 The n-queens problem 488
23.3 Querying with `for` expressions 491
23.4 Translation of `for` expressions 493
23.5 Going the other way 497
23.6 Generalizing `for` 498
23.7 Conclusion 500

24 Collections in Depth 501
24.1 Mutable and immutable collections 502
24.2 Collections consistency 504
24.3 Trait `Traversable` 506
24.4 Trait `Iterable` 511
24.5 The sequence traits `Seq`, `IndexedSeq`, and `LinearSeq` . 515
24.6 Sets 520
24.7 Maps 525

24.8 Concrete immutable collection classes 529
24.9 Concrete mutable collection classes 536
24.10 Arrays 543
24.11 Strings 548
24.12 Performance characteristics 548
24.13 Equality 551
24.14 Views 552
24.15 Iterators 557
24.16 Creating collections from scratch 565
24.17 Conversions between Java and Scala collections 567
24.18 Conclusion 569

25 The Architecture of Scala Collections **571**
25.1 Builders 571
25.2 Factoring out common operations 573
25.3 Integrating new collections 578
25.4 Conclusion 594

26 Extractors **595**
26.1 An example: extracting email addresses 595
26.2 Extractors 596
26.3 Patterns with zero or one variables 599
26.4 Variable argument extractors 601
26.5 Extractors and sequence patterns 604
26.6 Extractors versus case classes 605
26.7 Regular expressions 606
26.8 Conclusion 610

27 Annotations **611**
27.1 Why have annotations? 611
27.2 Syntax of annotations 612
27.3 Standard annotations 614
27.4 Conclusion 618

28 Working with XML **619**
28.1 Semi-structured data 619
28.2 XML overview 620
28.3 XML literals 621

28.4 Serialization . . . 623
28.5 Taking XML apart . . . 625
28.6 Deserialization . . . 626
28.7 Loading and saving . . . 627
28.8 Pattern matching on XML . . . 629
28.9 Conclusion . . . 632

29 Modular Programming Using Objects **633**
29.1 The problem . . . 634
29.2 A recipe application . . . 635
29.3 Abstraction . . . 638
29.4 Splitting modules into traits . . . 641
29.5 Runtime linking . . . 644
29.6 Tracking module instances . . . 645
29.7 Conclusion . . . 647

30 Object Equality **649**
30.1 Equality in Scala . . . 649
30.2 Writing an equality method . . . 650
30.3 Defining equality for parameterized types . . . 663
30.4 Recipes for `equals` and `hashCode` . . . 667
30.5 Conclusion . . . 672

31 Combining Scala and Java **673**
31.1 Using Scala from Java . . . 673
31.2 Annotations . . . 676
31.3 Wildcard types . . . 681
31.4 Compiling Scala and Java together . . . 684
31.5 Java 8 integration in Scala 2.12 . . . 684
31.6 Conclusion . . . 688

32 Futures and Concurrency **689**
32.1 Trouble in paradise . . . 689
32.2 Asynchronous execution and `Try`s . . . 691
32.3 Working with `Future`s . . . 693
32.4 Testing with `Future`s . . . 708
32.5 Conclusion . . . 710

33 Combinator Parsing **711**
33.1 Example: Arithmetic expressions 712
33.2 Running your parser 714
33.3 Basic regular expression parsers 715
33.4 Another example: JSON 716
33.5 Parser output 718
33.6 Implementing combinator parsers 724
33.7 String literals and regular expressions 733
33.8 Lexing and parsing 734
33.9 Error reporting 734
33.10 Backtracking versus LL(1) 736
33.11 Conclusion 738

34 GUI Programming **741**
34.1 A first Swing application 741
34.2 Panels and layouts 744
34.3 Handling events 746
34.4 Example: Celsius/Fahrenheit converter 749
34.5 Conclusion 751

35 The `SCells` Spreadsheet **753**
35.1 The visual framework 753
35.2 Disconnecting data entry and display 756
35.3 Formulas 759
35.4 Parsing formulas 761
35.5 Evaluation 766
35.6 Operation libraries 769
35.7 Change propagation 772
35.8 Conclusion 776

A Scala Scripts on Unix and Windows **779**

Glossary **781**

Bibliography **797**

About the Authors **801**

Index **803**