# *brief contents*


**PART 1 FUNDAMENTALS .....1**

1 ■ Java 8: why should you care? 3
2 ■ Passing code with behavior parameterization 24
3 ■ Lambda expressions 39

**PART 2 FUNCTIONAL-STYLE DATA PROCESSING .....75**

4 ■ Introducing streams 77
5 ■ Working with streams 92
6 ■ Collecting data with streams 123
7 ■ Parallel data processing and performance 158

**PART 3 EFFECTIVE JAVA 8 PROGRAMMING .....183**

8 ■ Refactoring, testing, and debugging 185
9 ■ Default methods 207
10 ■ Using Optional as a better alternative to null 225
11 ■ CompletableFuture: composable asynchronous programming 245
12 ■ New Date and Time API 273

PART 4 BEYOND JAVA 8 .....289

13 ■ Thinking functionally 291
14 ■ Functional programming techniques 305
15 ■ Blending OOP and FP: comparing Java 8 and Scala 329
16 ■ Conclusions and where next for Java 344

APPENDIXES .....358

A ■ Miscellaneous language updates 358
B ■ Miscellaneous library updates 362
C ■ Performing multiple operations in parallel on a stream 370
D ■ Lambdas and JVM bytecode 379

# *contents*

*preface xvii*
*acknowledgments xix*
*about this book xxi*
*about the authors xxv*
*about the cover illustration xxvii*

**PART 1 FUNDAMENTALS.....1**

*1* ***Java 8: why should you care? 3***

1.1 Why is Java still changing? 5

*Java's place in the programming language ecosystem 6 Stream processing 7 ▪ Passing code to methods with behavior parameterization 9 ▪ Parallelism and shared mutable data 9 ▪ Java needs to evolve 10*

1.2 Functions in Java 11

*Methods and lambdas as first-class citizens 12 ▪ Passing code: an example 13 ▪ From passing methods to lambdas 15*

1.3 Streams 16

*Multithreading is difficult 17*

1.4 Default methods 20

1.5 Other good ideas from functional programming 21

1.6 Summary 23

2 *Passing code with behavior parameterization 24*

2.1 Coping with changing requirements 25

*First attempt: filtering green apples 25 ▪ Second attempt: parameterizing the color 26 ▪ Third attempt: filtering with every attribute you can think of 27*

2.2 Behavior parameterization 27

*Fourth attempt: filtering by abstract criteria 29*

2.3 Tackling verbosity 32

*Anonymous classes 33 ▪ Fifth attempt: using an anonymous class 33 ▪ Sixth attempt: using a lambda expression 35 ▪ Seventh attempt: abstracting over List type 36*

2.4 Real-world examples 36

*Sorting with a Comparator 36 ▪ Executing a block of code with Runnable 37 ▪ GUI event handling 38*

2.5 Summary 38

3 *Lambda expressions 39*

3.1 Lambdas in a nutshell 40

3.2 Where and how to use lambdas 43

*Functional interface 43 ▪ Function descriptor 45*

3.3 Putting lambdas into practice: the execute around pattern 47

*Step 1: Remember behavior parameterization 47*
*Step 2: Use a functional interface to pass behaviors 48*
*Step 3: Execute a behavior! 48 ▪ Step 4: Pass lambdas 48*

3.4 Using functional interfaces 50

*Predicate 50 ▪ Consumer 50 ▪ Function 51*

3.5 Type checking, type inference, and restrictions 56

*Type checking 56 ▪ Same lambda, different functional interfaces 57 ▪ Type inference 58 ▪ Using local variables 59*

3.6 Method references 60

*In a nutshell 60 ▪ Constructor references 63*

3.7 Putting lambdas and method references into practice! 65

*Step 1: Pass code 65 ▪ Step 2: Use an anonymous class 66*
*Step 3: Use lambda expressions 66 ▪ Step 4: Use method references 67*

3.8 Useful methods to compose lambda expressions 67
*Composing Comparators 67 ▪ Composing Predicates 68*
*Composing Functions 68*
3.9 Similar ideas from mathematics 70
*Integration 70 ▪ Connecting to Java 8 lambdas 72*
3.10 Summary 72

PART 2 FUNCTIONAL-STYLE DATA PROCESSING .....75

4 *Introducing streams 77*
4.1 What are streams? 78
4.2 Getting started with streams 81
4.3 Streams vs. collections 84
*Traversable only once 86 ▪ External vs. internal iteration 86*
4.4 Stream operations 88
*Intermediate operations 89 ▪ Terminal operations 90*
*Working with streams 90*
4.5 Summary 91

5 *Working with streams 92*
5.1 Filtering and slicing 93
*Filtering with a predicate 93 ▪ Filtering unique elements 94*
*Truncating a stream 94 ▪ Skipping elements 95*
5.2 Mapping 96
*Applying a function to each element of a stream 96*
*Flattening streams 97*
5.3 Finding and matching 100
*Checking to see if a predicate matches at least one element 100*
*Checking to see if a predicate matches all elements 101*
*Finding an element 101 ▪ Finding the first element 102*
5.4 Reducing 103
*Summing the elements 103 ▪ Maximum and minimum 105*
5.5 Putting it all into practice 108
*The domain: Traders and Transactions 109 ▪ Solutions 110*
5.6 Numeric streams 112
*Primitive stream specializations 112 ▪ Numeric ranges 114*
*Putting numerical streams into practice: Pythagorean triples 114*

5.7 Building streams 117

*Streams from values 117 ▪ Streams from arrays 117*
*Streams from files 117 ▪ Streams from functions: creating infinite streams! 118*

5.8 Summary 121

6 *Collecting data with streams 123*

6.1 Collectors in a nutshell 125

*Collectors as advanced reductions 125*
*Predefined collectors 126*

6.2 Reducing and summarizing 126

*Finding maximum and minimum in a stream of values 127*
*Summarization 128 ▪ Joining Strings 129*
*Generalized summarization with reduction 130*

6.3 Grouping 134

*Multilevel grouping 135 ▪ Collecting data in subgroups 137*

6.4 Partitioning 140

*Advantages of partitioning 141 ▪ Partitioning numbers into prime and nonprime 142*

6.5 The Collector interface 145

*Making sense of the methods declared by Collector interface 146*
*Putting them all together 149*

6.6 Developing your own collector for better performance 151

*Divide only by prime numbers 151*
*Comparing collectors' performances 155*

6.7 Summary 156

7 *Parallel data processing and performance 158*

7.1 Parallel streams 159

*Turning a sequential stream into a parallel one 160*
*Measuring stream performance 162 ▪ Using parallel streams correctly 165 ▪ Using parallel streams effectively 166*

7.2 The fork/join framework 168

*Working with RecursiveTask 168 ▪ Best practices for using the fork/join framework 172 ▪ Work stealing 173*

7.3 Spliterator 174

*The splitting process 175 ▪ Implementing your own Spliterator 176*

7.4 Summary 182

PART 3 EFFECTIVE JAVA 8 PROGRAMMING .....183

8 *Refactoring, testing, and debugging 185*

8.1 Refactoring for improved readability and flexibility 186

*Improving code readability 186 ▪ From anonymous classes to lambda expressions 186 ▪ From lambda expressions to method references 188 ▪ From imperative data processing to Streams 189 ▪ Improving code flexibility 190*

8.2 Refactoring object-oriented design patterns with lambdas 192

*Strategy 192 ▪ Template method 194 ▪ Observer 195 Chain of responsibility 197 ▪ Factory 199*

8.3 Testing lambdas 200

*Testing the behavior of a visible lambda 201 Focusing on the behavior of the method using a lambda 201 Pulling complex lambdas into separate methods 202 Testing high-order functions 202*

8.4 Debugging 203

*Examining the stack trace 203 ▪ Logging information 205*

8.5 Summary 206

9 *Default methods 207*

9.1 Evolving APIs 210

*API version 1 210 ▪ API version 2 211*

9.2 Default methods in a nutshell 213

9.3 Usage patterns for default methods 215

*Optional methods 215 ▪ Multiple inheritance of behavior 215*

9.4 Resolution rules 219

*Three resolution rules to know 219 ▪ Most specific default-providing interface wins 220 ▪ Conflicts and explicit disambiguation 221 ▪ Diamond problem 223*

9.5 Summary 224

10 *Using Optional as a better alternative to null 225*

10.1 How do you model the absence of a value? 226

*Reducing NullPointerExceptions with defensive checking 227*
*Problems with null 228 ▪ What are the alternatives to null in other languages? 229*

10.2 Introducing the Optional class 230

10.3 Patterns for adopting Optional 231

*Creating Optional objects 231 ▪ Extracting and transforming values from optionals with map 232 ▪ Chaining Optional objects with flatMap 233 ▪ Default actions and unwrapping an optional 236 ▪ Combining two optionals 237*
*Rejecting certain values with filter 238*

10.4 Practical examples of using Optional 240

*Wrapping a potentially null value in an optional 240*
*Exceptions vs. Optional 241 ▪ Putting it all together 242*

10.5 Summary 243

11 *CompletableFuture: composable asynchronous programming 245*

11.1 Futures 247

*Futures limitations 248 ▪ Using CompletableFutures to build an asynchronous application 249*

11.2 Implementing an asynchronous API 250

*Converting a synchronous method into an asynchronous one 251*
*Dealing with errors 253*

11.3 Make your code non-blocking 254

*Parallelizing requests using a parallel Stream 255*
*Making asynchronous requests with CompletableFutures 256*
*Looking for the solution that scales better 258*
*Using a custom Executor 259*

11.4 Pipelining asynchronous tasks 261

*Implementing a discount service 262 ▪ Using the Discount service 263 ▪ Composing synchronous and asynchronous operations 264 ▪ Combining two CompletableFutures—dependent and independent 266*
*Reflecting on Future vs. CompletableFuture 267*

11.5 Reacting to a CompletableFuture completion 269

*Refactoring the best-price-finder application 269*
*Putting it to work 271*

11.6 Summary 272

12 ***New Date and Time API 273***

12.1 LocalDate, LocalTime, Instant, Duration, and Period 274

*Working with LocalDate and LocalTime 275 ▪ Combining a date and a time 276 ▪ Instant: a date and time for machines 276*
*Defining a Duration or a Period 277*

12.2 Manipulating, parsing, and formatting dates 279

*Working with TemporalAdjusters 280 ▪ Printing and parsing date-time objects 283*

12.3 Working with different time zones and calendars 285

*Fixed offset from UTC/Greenwich 286 ▪ Using alternative calendar systems 286*

12.4 Summary 287

**PART 4 BEYOND JAVA 8 ..... 289**

13 ***Thinking functionally 291***

13.1 Implementing and maintaining systems 292

*Shared mutable data 292 ▪ Declarative programming 293*
*Why functional programming? 294*

13.2 What's functional programming? 294

*Functional-style Java 295 ▪ Referential transparency 297*
*Object-oriented vs. functional-style programming 298*
*Functional style in practice 298*

13.3 Recursion vs. iteration 300

13.4 Summary 304

14 ***Functional programming techniques 305***

14.1 Functions everywhere 306

*Higher-order functions 306 ▪ Currying 307*

14.2 Persistent data structures 309

*Destructive updates vs. functional 309 ▪ Another example with Trees 310 ▪ Using a functional approach 312*

14.3 Lazy evaluation with streams 314
*Self-defining stream 314 ▪ Your own lazy list 317*
14.4 Pattern matching 321
*Visitor design pattern 322 ▪ Pattern matching to the rescue 322*
14.5 Miscellany 325
*Caching or memoization 325 ▪ What does "return the same object" mean? 327 ▪ Combinators 327*
14.6 Summary 328

15 ***Blending OOP and FP: comparing Java 8 and Scala 329***
15.1 Introduction to Scala 330
*Hello beer 330 ▪ Basic data structures: List, Set, Map, Tuple, Stream, Option 332*
15.2 Functions 337
*First-class functions in Scala 337 ▪ Anonymous functions and closures 338 ▪ Currying 339*
15.3 Classes and traits 341
*Less verbosity with Scala classes 341 ▪ Scala traits vs. Java 8 interfaces 342*
15.4 Summary 343

16 ***Conclusions and where next for Java 344***
16.1 Review of Java 8 features 344
*Behavior parameterization (lambdas and method references) 345 Streams 346 ▪ CompletableFuture 346 ▪ Optional 347 Default methods 347*
16.2 What's ahead for Java? 348
*Collections 348 ▪ Type system enhancements 348 Pattern matching 350 ▪ Richer forms of generics 351 Deeper support for immutability 353 ▪ Value types 353*
16.3 The final word 357

*appendix A Miscellaneous language updates 358*
*appendix B Miscellaneous library updates 362*
*appendix C Performing multiple operations in parallel on a stream 370*
*appendix D Lambdas and JVM bytecode 379*

*index 385*