contents

foreword xi
preface xiii
acknowledgments xiv
about this book xvi
about the cover illustration xix

1 Scala—a blended language 1

1.1 Functional programming meets object orientation 2

Discovering existing functional concepts 4 ▪ Examining functional concepts in Google Collections 6

1.2 Static typing and expressiveness 8

Changing sides 8 ▪ Type inference 9 ▪ Dropping verbose syntax 9 ▪ Implicits are an old concept 10 ▪ Using Scala's implicit keyword 11

1.3 Transparently working with the JVM 12

Java in Scala 12 ▪ Scala in Java 13 ▪ The benefits of a JVM 14

1.4 Summary 15

2 ***The core rules* 16**

2.1 Learn to use the Read Eval Print Loop (REPL) 16

Experiment-driven development 18 ▪ Working around eager parsing 19 ▪ Inexpressible language features 20

2.2 Think in expressions 21

Don't use return 22 ▪ Mutability 24

2.3 Prefer immutability 26

Object equality 27 ▪ Concurrency 31

2.4 Use None instead of null 34

Advanced Option techniques 35

2.5 Polymorphic equality 38

Example: A timeline library 38 ▪ Polymorphic equals implementation 40

2.6 Summary 42

3 ***Modicum of style—coding conventions* 43**

3.1 Avoid coding conventions from other languages 44

The block debacle 45

3.2 Dangling operators and parenthetical expressions 48

3.3 Use meaningful variable names 49

Avoid $ in names 50 ▪ Working with named and default parameters 53

3.4 Always mark overridden methods 55

3.5 Annotate for expected optimizations 60

Using the tableswitch optimization 61 ▪ Using the tail recursion optimization 64

3.6 Summary 66

4 ***Utilizing object orientation* 68**

4.1 Limit code inside an object or trait's body to initialization logic 69

Delayed construction 69 ▪ And then there's multiple inheritance 70

4.2 Provide empty implementations for abstract methods on traits 72

4.3 Composition can include inheritance 76

Member composition by inheritance 78 ▪ Classic constructors with a twist 80

4.4 Promote abstract interface into its own trait 82

Interfaces you can talk to 84 ▪ Learning from the past 85

4.5 Provide return types in your public APIs 86

4.6 Summary 88

5 *Using implicits to write expressive code 89*

5.1 Introduction to implicits 90

Identifiers: A digression 91 ▪ Scope and bindings 92 Implicit resolution 96

5.2 Enhancing existing classes with implicit views 101

5.3 Utilize implicit parameters with defaults 106

5.4 Limiting the scope of implicits 112

Creating implicits for import 113 ▪ Implicits without the import tax 115

5.5 Summary 119

6 *The type system 120*

6.1 Types 121

Types and paths 122 ▪ The type keyword 124 ▪ Structural types 125

6.2 Type constraints 131

6.3 Type parameters and higher-kinded types 134

Type parameter constraints 134 ▪ Higher-kinded types 135

6.4 Variance 137

Advanced variance annotations 141

6.5 Existential types 144

The formal syntax of existential types 146

6.6 Summary 149

7 *Using implicits and types together 150*

7.1 Context bounds and view bounds 151

When to use implicit type constraints 152

7.2 Capturing types with implicits 153

Manifests 153 ▪ Using Manifests 154 ▪ Capturing type constraints 156 ▪ Specialized methods 158

7.3 Use type classes 159

FileLike as a type class 163 ▪ The benefits of type classes 166

7.4 Conditional execution using the type system 167

Heterogeneous typed list 169 ▪ IndexedView 172

7.5 Summary 178

8 **Using the right collection 179**

8.1 Use the right collection 180

The collection hierarchy 180 ▪ Traversable 182
Iterable 185 ▪ Seq 187 ▪ LinearSeq 187
IndexedSeq 189 ▪ Set 190 ▪ Map 191

8.2 Immutable collections 192

Vector 192 ▪ List 194 ▪ Stream 195

8.3 Mutable collections 198

ArrayBuffer 198 ▪ Mixin mutation event publishing 199 ▪ Mixin synchronization 200

8.4 Changing evaluation with views and parallel collections 200

Views 201 ▪ Parallel collections 203

8.5 Writing methods to use with all collection types 205

Optimizing algorithms for each collections type 209

8.6 Summary 211

9 **Actors 212**

9.1 Know when to use actors 213

Using actors to search 213

9.2 Use typed, transparent references 216

Scatter-Gather with OutputChannel 217

9.3 Limit failures to zones 221

Scatter-Gather failure zones 221 ▪ General failure handling practices 224

9.4 Limit overload using scheduler zones 225

Scheduling zones 227

9.5 Dynamic actor topology 228
9.6 Summary 233

10 *Integrating Scala with Java 234*

10.1 The language mismatch between Scala and Java 235
Differences in primitive boxing 236 ▪ Differences in visibility 240 ▪ Inexpressible language features 241
10.2 Be wary of implicit conversions 244
Object identity and equality 245 ▪ Chaining implicits 246
10.3 Be wary of Java serialization 248
Serializing anonymous classes 250
10.4 Annotate your annotations 252
Annotation targets 254 ▪ Scala and static fields 255
10.5 Summary 256

11 *Patterns in functional programming 257*

11.1 Category theory for computer science 258
11.2 Functors and monads, and how they relate to categories 262
Monads 264
11.3 Currying and applicative style 266
Currying 266 ▪ Applicative style 268
11.4 Monads as workflows 272
11.5 Summary 276

index 277