

# Contents

<b>Contents</b>	<b>ix</b>
<b>List of Figures</b>	<b>xix</b>
<b>List of Tables</b>	<b>xxi</b>
<b>List of Listings</b>	<b>xxii</b>
<b>Foreword</b>	<b>xxix</b>
<b>Acknowledgments</b>	<b>xxxii</b>
<b>Introduction</b>	<b>xxxiii</b>
<b>1 A Scalable Language</b>	<b>3</b>
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
<b>2 First Steps in Scala</b>	<b>23</b>
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 <code>while</code> ; decide with <code>if</code> . . . . .	30
Step 6. Iterate with <code>foreach</code> and <code>for</code> . . . . .	32
Conclusion . . . . .	35

<b>3</b>	<b>Next Steps in Scala</b>	<b>37</b>
	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
<b>4</b>	<b>Classes and Objects</b>	<b>59</b>
	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 Application trait . . . . .	71
	4.6 Conclusion . . . . .	72
<b>5</b>	<b>Basic Types and Operations</b>	<b>73</b>
	5.1 Some basic types . . . . .	73
	5.2 Literals . . . . .	74
	5.3 Operators are methods . . . . .	81
	5.4 Arithmetic operations . . . . .	84
	5.5 Relational and logical operations . . . . .	85
	5.6 Bitwise operations . . . . .	87
	5.7 Object equality . . . . .	89
	5.8 Operator precedence and associativity . . . . .	90
	5.9 Rich wrappers . . . . .	93
	5.10 Conclusion . . . . .	93
<b>6</b>	<b>Functional Objects</b>	<b>95</b>
	6.1 A specification for class Rational . . . . .	95
	6.2 Constructing a Rational . . . . .	96
	6.3 Reimplementing the toString method . . . . .	98
	6.4 Checking preconditions . . . . .	99
	6.5 Adding fields . . . . .	99
	6.6 Self references . . . . .	101
	6.7 Auxiliary constructors . . . . .	102
	6.8 Private fields and methods . . . . .	104

6.9	Defining operators . . . . .	105
6.10	Identifiers in Scala . . . . .	107
6.11	Method overloading . . . . .	110
6.12	Implicit conversions . . . . .	112
6.13	A word of caution . . . . .	113
6.14	Conclusion . . . . .	113
<b>7</b>	<b>Built-in Control Structures</b>	<b>115</b>
7.1	If expressions . . . . .	116
7.2	While loops . . . . .	117
7.3	For expressions . . . . .	120
7.4	Exception handling with try expressions . . . . .	125
7.5	Match expressions . . . . .	129
7.6	Living without break and continue . . . . .	131
7.7	Variable scope . . . . .	133
7.8	Refactoring imperative-style code . . . . .	136
7.9	Conclusion . . . . .	138
<b>8</b>	<b>Functions and Closures</b>	<b>139</b>
8.1	Methods . . . . .	139
8.2	Local functions . . . . .	141
8.3	First-class functions . . . . .	143
8.4	Short forms of function literals . . . . .	145
8.5	Placeholder syntax . . . . .	146
8.6	Partially applied functions . . . . .	147
8.7	Closures . . . . .	150
8.8	Repeated parameters . . . . .	154
8.9	Tail recursion . . . . .	155
8.10	Conclusion . . . . .	159
<b>9</b>	<b>Control Abstraction</b>	<b>161</b>
9.1	Reducing code duplication . . . . .	161
9.2	Simplifying client code . . . . .	165
9.3	Currying . . . . .	167
9.4	Writing new control structures . . . . .	169
9.5	By-name parameters . . . . .	172
9.6	Conclusion . . . . .	175

<b>10</b>	<b>Composition and Inheritance</b>	<b>177</b>
10.1	A two-dimensional layout library . . . . .	177
10.2	Abstract classes . . . . .	178
10.3	Defining parameterless methods . . . . .	179
10.4	Extending classes . . . . .	182
10.5	Overriding methods and fields . . . . .	184
10.6	Defining parametric fields . . . . .	185
10.7	Invoking superclass constructors . . . . .	187
10.8	Using override modifiers . . . . .	188
10.9	Polymorphism and dynamic binding . . . . .	190
10.10	Declaring final members . . . . .	192
10.11	Using composition and inheritance . . . . .	194
10.12	Implementing above, beside, and toString . . . . .	195
10.13	Defining a factory object . . . . .	197
10.14	Heighten and widen . . . . .	199
10.15	Putting it all together . . . . .	203
10.16	Conclusion . . . . .	204
<b>11</b>	<b>Scala's Hierarchy</b>	<b>205</b>
11.1	Scala's class hierarchy . . . . .	205
11.2	How primitives are implemented . . . . .	209
11.3	Bottom types . . . . .	211
11.4	Conclusion . . . . .	212
<b>12</b>	<b>Traits</b>	<b>213</b>
12.1	How traits work . . . . .	213
12.2	Thin versus rich interfaces . . . . .	216
12.3	Example: Rectangular objects . . . . .	217
12.4	The Ordered trait . . . . .	220
12.5	Traits as stackable modifications . . . . .	222
12.6	Why not multiple inheritance? . . . . .	226
12.7	To trait, or not to trait? . . . . .	230
12.8	Conclusion . . . . .	231
<b>13</b>	<b>Packages and Imports</b>	<b>233</b>
13.1	Packages . . . . .	233
13.2	Imports . . . . .	237
13.3	Implicit imports . . . . .	241

13.4	Access modifiers . . . . .	242
13.5	Conclusion . . . . .	247
<b>14</b>	<b>Assertions and Unit Testing</b>	<b>249</b>
14.1	Assertions . . . . .	249
14.2	Unit testing in Scala . . . . .	251
14.3	Informative failure reports . . . . .	252
14.4	Using JUnit and TestNG . . . . .	254
14.5	Tests as specifications . . . . .	256
14.6	Property-based testing . . . . .	258
14.7	Organizing and running tests . . . . .	260
14.8	Conclusion . . . . .	262
<b>15</b>	<b>Case Classes and Pattern Matching</b>	<b>263</b>
15.1	A simple example . . . . .	263
15.2	Kinds of patterns . . . . .	268
15.3	Pattern guards . . . . .	277
15.4	Pattern overlaps . . . . .	279
15.5	Sealed classes . . . . .	280
15.6	The Option type . . . . .	282
15.7	Patterns everywhere . . . . .	284
15.8	A larger example . . . . .	288
15.9	Conclusion . . . . .	296
<b>16</b>	<b>Working with Lists</b>	<b>297</b>
16.1	List literals . . . . .	297
16.2	The List type . . . . .	298
16.3	Constructing lists . . . . .	298
16.4	Basic operations on lists . . . . .	299
16.5	List patterns . . . . .	300
16.6	First-order methods on class List . . . . .	302
16.7	Higher-order methods on class List . . . . .	313
16.8	Methods of the List object . . . . .	321
16.9	Understanding Scala's type inference algorithm . . . . .	325
16.10	Conclusion . . . . .	328
<b>17</b>	<b>Collections</b>	<b>329</b>
17.1	Overview of the library . . . . .	329

17.2	Sequences . . . . .	331
17.3	Sets and maps . . . . .	337
17.4	Selecting mutable versus immutable collections . . . . .	347
17.5	Initializing collections . . . . .	350
17.6	Tuples . . . . .	353
17.7	Conclusion . . . . .	356
<b>18</b>	<b>Stateful Objects</b>	<b>357</b>
18.1	What makes an object stateful? . . . . .	357
18.2	Reassignable variables and properties . . . . .	360
18.3	Case study: Discrete event simulation . . . . .	363
18.4	A language for digital circuits . . . . .	364
18.5	The Simulation API . . . . .	367
18.6	Circuit Simulation . . . . .	371
18.7	Conclusion . . . . .	379
<b>19</b>	<b>Type Parameterization</b>	<b>381</b>
19.1	Functional queues . . . . .	381
19.2	Information hiding . . . . .	385
19.3	Variance annotations . . . . .	388
19.4	Checking variance annotations . . . . .	392
19.5	Lower bounds . . . . .	395
19.6	Contravariance . . . . .	397
19.7	Object private data . . . . .	400
19.8	Upper bounds . . . . .	402
19.9	Conclusion . . . . .	405
<b>20</b>	<b>Abstract Members</b>	<b>407</b>
20.1	A quick tour of abstract members . . . . .	407
20.2	Type members . . . . .	408
20.3	Abstract vals . . . . .	409
20.4	Abstract vars . . . . .	410
20.5	Initializing abstract vals . . . . .	411
20.6	Abstract types . . . . .	419
20.7	Path-dependent types . . . . .	421
20.8	Enumerations . . . . .	424
20.9	Case study: Currencies . . . . .	426
20.10	Conclusion . . . . .	435

<b>21</b>	<b>Implicit Conversions and Parameters</b>	<b>437</b>
21.1	Implicit conversions . . . . .	437
21.2	Rules for implicits . . . . .	440
21.3	Implicit conversion to an expected type . . . . .	443
21.4	Converting the receiver . . . . .	445
21.5	Implicit parameters . . . . .	447
21.6	View bounds . . . . .	453
21.7	Debugging implicits . . . . .	457
21.8	Conclusion . . . . .	458
<b>22</b>	<b>Implementing Lists</b>	<b>459</b>
22.1	The List class in principle . . . . .	459
22.2	The ListBuffer class . . . . .	465
22.3	The List class in practice . . . . .	467
22.4	Functional on the outside . . . . .	469
22.5	Conclusion . . . . .	470
<b>23</b>	<b>For Expressions Revisited</b>	<b>473</b>
23.1	For expressions . . . . .	474
23.2	The n-queens problem . . . . .	476
23.3	Querying with for expressions . . . . .	479
23.4	Translation of for expressions . . . . .	481
23.5	Going the other way . . . . .	485
23.6	Generalizing for . . . . .	486
23.7	Conclusion . . . . .	488
<b>24</b>	<b>Extractors</b>	<b>489</b>
24.1	An example: Extracting email addresses . . . . .	489
24.2	Extractors . . . . .	490
24.3	Patterns with zero or one variables . . . . .	493
24.4	Variable argument extractors . . . . .	495
24.5	Extractors and sequence patterns . . . . .	498
24.6	Extractors versus case classes . . . . .	499
24.7	Regular expressions . . . . .	500
24.8	Conclusion . . . . .	504
<b>25</b>	<b>Annotations</b>	<b>505</b>
25.1	Why have annotations? . . . . .	505

25.2	Syntax of annotations . . . . .	506
25.3	Standard annotations . . . . .	508
25.4	Conclusion . . . . .	510
<b>26</b>	<b>Working with XML</b>	<b>513</b>
26.1	Semi-structured data . . . . .	513
26.2	XML overview . . . . .	514
26.3	XML literals . . . . .	515
26.4	Serialization . . . . .	517
26.5	Taking XML apart . . . . .	519
26.6	Deserialization . . . . .	520
26.7	Loading and saving . . . . .	521
26.8	Pattern matching on XML . . . . .	523
26.9	Conclusion . . . . .	526
<b>27</b>	<b>Modular Programming Using Objects</b>	<b>527</b>
27.1	The problem . . . . .	528
27.2	A recipe application . . . . .	529
27.3	Abstraction . . . . .	532
27.4	Splitting modules into traits . . . . .	535
27.5	Runtime linking . . . . .	538
27.6	Tracking module instances . . . . .	539
27.7	Conclusion . . . . .	541
<b>28</b>	<b>Object Equality</b>	<b>543</b>
28.1	Equality in Scala . . . . .	543
28.2	Writing an equality method . . . . .	544
28.3	Defining equality for parameterized types . . . . .	557
28.4	Recipes for equals and hashCode . . . . .	562
28.5	Conclusion . . . . .	568
<b>29</b>	<b>Combining Scala and Java</b>	<b>569</b>
29.1	Using Scala from Java . . . . .	569
29.2	Annotations . . . . .	572
29.3	Existential types . . . . .	577
29.4	Conclusion . . . . .	581
<b>30</b>	<b>Actors and Concurrency</b>	<b>583</b>



30.1	Trouble in paradise . . . . .	583
30.2	Actors and message passing . . . . .	584
30.3	Treating native threads as actors . . . . .	588
30.4	Better performance through thread reuse . . . . .	589
30.5	Good actors style . . . . .	592
30.6	A longer example: Parallel discrete event simulation . . . . .	599
30.7	Conclusion . . . . .	616
<b>31</b>	<b>Combinator Parsing</b>	<b>619</b>
31.1	Example: Arithmetic expressions . . . . .	620
31.2	Running your parser . . . . .	622
31.3	Basic regular expression parsers . . . . .	623
31.4	Another example: JSON . . . . .	624
31.5	Parser output . . . . .	626
31.6	Implementing combinator parsers . . . . .	632
31.7	String literals and regular expressions . . . . .	641
31.8	Lexing and parsing . . . . .	642
31.9	Error reporting . . . . .	642
31.10	Backtracking versus LL(1) . . . . .	644
31.11	Conclusion . . . . .	646
<b>32</b>	<b>GUI Programming</b>	<b>649</b>
32.1	A first Swing application . . . . .	649
32.2	Panels and layouts . . . . .	652
32.3	Handling events . . . . .	654
32.4	Example: Celsius/Fahrenheit converter . . . . .	657
32.5	Conclusion . . . . .	659
<b>33</b>	<b>The SCells Spreadsheet</b>	<b>661</b>
33.1	The visual framework . . . . .	661
33.2	Disconnecting data entry and display . . . . .	664
33.3	Formulas . . . . .	667
33.4	Parsing formulas . . . . .	669
33.5	Evaluation . . . . .	674
33.6	Operation libraries . . . . .	677
33.7	Change propagation . . . . .	680
33.8	Conclusion . . . . .	684

<b>A Scala scripts on Unix and Windows</b>	<b>687</b>
<b>Glossary</b>	<b>689</b>
<b>Bibliography</b>	<b>705</b>
<b>About the Authors</b>	<b>709</b>
<b>Index</b>	<b>711</b>

# List of Figures

2.1	The basic form of a function definition in Scala. . . . .	28
2.2	The syntax of a function literal in Scala. . . . .	34
3.1	All operations are method calls in Scala. . . . .	40
3.2	Class hierarchy for Scala sets. . . . .	48
3.3	Class hierarchy for Scala maps. . . . .	50
10.1	Class diagram for <code>ArrayElement</code> . . . . .	183
10.2	Class diagram for <code>LineElement</code> . . . . .	188
10.3	Class hierarchy of layout elements . . . . .	191
10.4	Class hierarchy with revised <code>LineElement</code> . . . . .	195
11.1	Class hierarchy of Scala. . . . .	207
12.1	Inheritance hierarchy and linearization of class <code>Cat</code> . . . . .	229
14.1	<code>ScalaTest</code> 's graphical reporter. . . . .	261
17.1	Class hierarchy for Scala collections. . . . .	330
17.2	Class hierarchy for <code>Iterator</code> . . . . .	331
18.1	Basic gates. . . . .	364
18.2	A half-adder circuit. . . . .	366
18.3	A full-adder circuit. . . . .	367
19.1	Covariance and contravariance in function type parameters. . . . .	400
22.1	Class hierarchy for Scala lists. . . . .	460
22.2	The structure of the Scala lists shown in Listing 22.2. . . . .	464

## List of Figures

32.1	A simple Swing application: initial (left) and resized (right).	650
32.2	A reactive Swing application: initial (left) after clicks (right).	652
32.3	A converter between degrees Celsius and Fahrenheit. . . . .	657
33.1	A simple spreadsheet table. . . . .	662
33.2	Cells displaying themselves. . . . .	667
33.3	Cells displaying their formulas. . . . .	673
33.4	Cells that evaluate. . . . .	679

# List of Tables

3.1	Some List methods and usages . . . . .	44
5.1	Some basic types . . . . .	74
5.2	Special character literal escape sequences . . . . .	78
5.3	Operator precedence . . . . .	92
5.4	Some rich operations . . . . .	94
5.5	Rich wrapper classes . . . . .	94
12.1	Linearization of types in Cat's hierarchy . . . . .	230
13.1	Effects of private qualifiers on LegOfJourney.distance . .	245
16.1	Basic list operations . . . . .	300
17.1	Common operations for sets . . . . .	340
17.2	Common operations for maps . . . . .	342
17.3	Default immutable set implementations . . . . .	344
17.4	Default immutable map implementations . . . . .	344
31.1	Summary of parser combinators . . . . .	630

# List of Listings

3.1	Parameterizing an array with a type. . . . .	38
3.2	Creating and initializing an array. . . . .	41
3.3	Creating and initializing a list. . . . .	42
3.4	Creating and using a tuple. . . . .	46
3.5	Creating, initializing, and using an immutable set. . . . .	47
3.6	Creating, initializing, and using a mutable set. . . . .	49
3.7	Creating, initializing, and using a mutable map. . . . .	50
3.8	Creating, initializing, and using an immutable map. . . . .	51
3.9	A function without side effects or vars. . . . .	53
3.10	Reading lines from a file. . . . .	55
3.11	Printing formatted character counts for the lines of a file. . . . .	58
4.1	Final version of class <code>ChecksumAccumulator</code> . . . . .	63
4.2	Companion object for class <code>ChecksumAccumulator</code> . . . . .	66
4.3	The <code>Summer</code> application. . . . .	68
4.4	Using the <code>Application</code> trait. . . . .	71
6.1	<code>Rational</code> with fields. . . . .	101
6.2	<code>Rational</code> with an auxiliary constructor. . . . .	103
6.3	<code>Rational</code> with a private field and method. . . . .	104
6.4	<code>Rational</code> with operator methods. . . . .	106
6.5	<code>Rational</code> with overloaded methods. . . . .	111
7.1	Scala’s idiom for conditional initialization. . . . .	116
7.2	Calculating greatest common divisor with a <code>while</code> loop. . . . .	117
7.3	Reading from the standard input with <code>do-while</code> . . . . .	118
7.4	Calculating greatest common divisor with recursion. . . . .	119
7.5	Listing files in a directory with a <code>for</code> expression. . . . .	120

7.6	Finding <code>.scala</code> files using a <code>for</code> with a filter. . . . .	122
7.7	Using multiple filters in a <code>for</code> expression. . . . .	122
7.8	Using multiple generators in a <code>for</code> expression. . . . .	123
7.9	Mid-stream assignment in a <code>for</code> expression. . . . .	124
7.10	Transforming an <code>Array[File]</code> to <code>Array[Int]</code> with a <code>for</code> . . . . .	125
7.11	A <code>try-catch</code> clause in Scala. . . . .	127
7.12	A <code>try-finally</code> clause in Scala. . . . .	128
7.13	A <code>catch</code> clause that yields a value. . . . .	129
7.14	A <code>match</code> expression with side effects. . . . .	130
7.15	A <code>match</code> expression that yields a value. . . . .	130
7.16	Looping without <code>break</code> or <code>continue</code> . . . . .	132
7.17	A recursive alternative to looping with <code>vars</code> . . . . .	132
7.18	Variable scoping when printing a multiplication table. . . . .	134
7.19	A functional way to create a multiplication table. . . . .	137
8.1	<code>LongLines</code> with a private <code>processLine</code> method. . . . .	140
8.2	<code>LongLines</code> with a local <code>processLine</code> function. . . . .	142
9.1	Using closures to reduce code duplication. . . . .	165
9.2	Defining and invoking a “plain old” function. . . . .	168
9.3	Defining and invoking a curried function. . . . .	168
9.4	Using the loan pattern to write to a file. . . . .	172
9.5	Using a by-name parameter. . . . .	173
10.1	Defining an abstract method and class. . . . .	179
10.2	Defining parameterless methods <code>width</code> and <code>height</code> . . . . .	180
10.3	Defining <code>ArrayElement</code> as a subclass of <code>Element</code> . . . . .	182
10.4	Overriding a parameterless method with a field. . . . .	184
10.5	Defining <code>contents</code> as a parametric field. . . . .	186
10.6	Invoking a superclass constructor. . . . .	187
10.7	Declaring a final method. . . . .	193
10.8	Declaring a final class. . . . .	193
10.9	Class <code>Element</code> with <code>above</code> , <code>beside</code> , and <code>toString</code> . . . . .	198
10.10	A factory object with factory methods. . . . .	199
10.11	Class <code>Element</code> refactored to use factory methods. . . . .	200
10.12	Hiding implementation with private classes. . . . .	201
10.13	<code>Element</code> with <code>widen</code> and <code>heighten</code> methods. . . . .	202
10.14	The <code>Spiral</code> application. . . . .	203

12.1	The definition of trait <code>Philosophical</code> . . . . .	213
12.2	Mixing in a trait using <code>extends</code> . . . . .	214
12.3	Mixing in a trait using <code>with</code> . . . . .	215
12.4	Mixing in multiple traits. . . . .	215
12.5	Defining an enrichment trait. . . . .	219
12.6	Abstract class <code>IntQueue</code> . . . . .	223
12.7	A <code>BasicIntQueue</code> implemented with an <code>ArrayBuffer</code> . . .	223
12.8	The <code>Doubling</code> stackable modification trait. . . . .	224
12.9	Mixing in a trait when instantiating with <code>new</code> . . . . .	225
12.10	Stackable modification traits <code>Incrementing</code> and <code>Filtering</code> . .	225
13.1	Placing the contents of an entire file into a package. . . . .	234
13.2	Nesting multiple packages in the same file. . . . .	234
13.3	Nesting packages with minimal indentation. . . . .	235
13.4	Scala packages truly nest. . . . .	236
13.5	Accessing hidden package names. . . . .	236
13.6	Bob’s delightful fruits, ready for import. . . . .	237
13.7	Importing the members of a regular (not singleton) object. .	238
13.8	Importing a package name. . . . .	239
13.9	How private access differs in Scala and Java. . . . .	242
13.10	How protected access differs in Scala and Java. . . . .	243
13.11	Flexible scope of protection with access qualifiers. . . . .	244
13.12	Accessing private members of companion classes and objects. .	246
14.1	Using an assertion. . . . .	250
14.2	Using <code>ensuring</code> to assert a function’s result. . . . .	250
14.3	Writing a test method with <code>Suite</code> . . . . .	251
14.4	Writing a test function with <code>FunSuite</code> . . . . .	252
14.5	Writing a JUnit test with <code>JUnit3Suite</code> . . . . .	255
14.6	Writing a TestNG test with <code>TestNGSuite</code> . . . . .	256
14.7	Specifying and testing behavior with a <code>ScalaTest Spec</code> . . .	257
14.8	Specifying and testing behavior with the specs framework. .	258
14.9	Writing property-based tests with <code>ScalaCheck</code> . . . . .	259
14.10	Checking properties from a JUnit <code>TestCase</code> with <code>Checkers</code> . .	260
15.1	Defining case classes. . . . .	264
15.2	The <code>simplifyTop</code> function, which does a pattern match. . .	266
15.3	A pattern match with an empty “default” case. . . . .	267



15.4	A pattern match with wildcard patterns. . . . .	268
15.5	A pattern match with constant patterns. . . . .	269
15.6	A pattern match with a variable pattern. . . . .	270
15.7	A pattern match with a constructor pattern. . . . .	272
15.8	A sequence pattern with a fixed length. . . . .	272
15.9	A sequence pattern with an arbitrary length. . . . .	273
15.10	A pattern match with a tuple pattern. . . . .	273
15.11	A pattern match with typed patterns. . . . .	273
15.12	Using <code>isInstanceOf</code> and <code>asInstanceOf</code> (poor style). . . . .	275
15.13	A pattern with a variable binding (via the <code>@ sign</code> ). . . . .	277
15.14	A match expression with a pattern guard. . . . .	278
15.15	Match expression in which case order matters. . . . .	279
15.16	A sealed hierarchy of case classes. . . . .	281
15.17	Defining multiple variables with one assignment. . . . .	284
15.18	A for expression with a tuple pattern. . . . .	288
15.19	Picking elements of a list that match a pattern. . . . .	288
15.20	The top half of the expression formatter. . . . .	291
15.21	The bottom half of the expression formatter. . . . .	292
15.22	An application that prints formatted expressions. . . . .	295
16.1	A merge sort function for Lists. . . . .	312
17.1	Default map and set definitions in Predef. . . . .	338
17.2	Mixing in the SynchronizedMap trait. . . . .	346
18.1	A mutable bank account class. . . . .	358
18.2	A class with public vars. . . . .	360
18.3	How public vars are expanded into getter and setter methods. . . . .	361
18.4	Defining getter and setter methods directly. . . . .	361
18.5	Defining a getter and setter without an associated field. . . . .	362
18.6	The <code>halfAdder</code> method. . . . .	365
18.7	The <code>fullAdder</code> method. . . . .	366
18.8	The <code>Simulation</code> class. . . . .	368
18.9	The first half of the <code>BasicCircuitSimulation</code> class. . . . .	372
18.10	The second half of the <code>BasicCircuitSimulation</code> class. . . . .	373
18.11	The <code>CircuitSimulation</code> class. . . . .	377
19.1	A basic functional queue. . . . .	384
19.2	Hiding a primary constructor by making it private. . . . .	385

19.3	An <code>apply</code> factory method in a companion object. . . . .	386
19.4	Type abstraction for functional queues. . . . .	387
19.5	A nonvariant (rigid) <code>Cell</code> class. . . . .	390
19.6	A type parameter with a lower bound. . . . .	396
19.7	A contravariant output channel. . . . .	397
19.8	Covariance and contravariance of <code>Function1</code> s. . . . .	398
19.9	Demonstration of function type parameter variance. . . . .	399
19.10	An optimized functional queue. . . . .	401
19.11	A <code>Person</code> class that mixes in the <code>Ordered</code> trait. . . . .	403
19.12	A merge sort function with an upper bound. . . . .	403
20.1	Overriding abstract <code>vals</code> and parameterless methods. . . . .	410
20.2	Declaring abstract vars. . . . .	410
20.3	How abstract vars are expanded into getters and setters. . . . .	411
20.4	A trait that uses its abstract <code>vals</code> . . . . .	412
20.5	Pre-initialized fields in an anonymous class expression. . . . .	414
20.6	Pre-initialized fields in an object definition. . . . .	414
20.7	Pre-initialized fields in a class definition. . . . .	415
20.8	Initializing a trait with lazy <code>vals</code> . . . . .	416
20.9	Modeling suitable food with an abstract type. . . . .	420
20.10	Implementing an abstract type in a subclass. . . . .	421
20.11	The US currency zone. . . . .	431
20.12	Currency zones for Europe and Japan. . . . .	432
20.13	A converter object with an exchange rates map. . . . .	433
20.14	The full code of class <code>CurrencyZone</code> . . . . .	434
21.1	An implicit parameter list with multiple parameters. . . . .	449
21.2	A function with an upper bound. . . . .	451
21.3	A function with an implicit parameter. . . . .	452
21.4	A function that uses an implicit parameter internally. . . . .	454
21.5	A function with a view bound. . . . .	455
21.6	Sample code that uses an implicit parameter. . . . .	456
21.7	Sample code after type checking and insertion of implicits. . . . .	456
22.1	The definition of the <code>Nil</code> singleton object. . . . .	461
22.2	Prepending a supertype element to a subtype list. . . . .	463
22.3	The definition of method <code>::</code> ( <code>cons</code> ) in class <code>List</code> . . . . .	463
22.4	The definition of method <code>:::</code> in class <code>List</code> . . . . .	465

22.5	The definition of method map in class <code>List</code> . . . . .	467
22.6	The definition of the <code>::</code> subclass of <code>List</code> . . . . .	468
24.1	The <code>EMail</code> string extractor object. . . . .	491
24.2	The <code>Twice</code> string extractor object. . . . .	494
24.3	The <code>UpperCase</code> string extractor object. . . . .	494
24.4	The <code>Domain</code> string extractor object. . . . .	496
24.5	The <code>ExpandedEMail</code> extractor object. . . . .	497
24.6	Hiding a primary constructor by making it private. . . . .	498
24.7	How the <code>r</code> method is defined in <code>RichString</code> . . . . .	502
27.1	A simple <code>Food</code> entity class. . . . .	529
27.2	Simple <code>Recipe</code> entity class. . . . .	530
27.3	<code>Food</code> and <code>Recipe</code> examples for use in tests. . . . .	530
27.4	Mock database and browser modules. . . . .	531
27.5	Database and browser modules with categories added. . . . .	532
27.6	A <code>Browser</code> class with an abstract database val. . . . .	533
27.7	A <code>Database</code> class with abstract methods. . . . .	534
27.8	The <code>SimpleDatabase</code> object as a <code>Database</code> subclass. . . . .	534
27.9	The <code>SimpleBrowser</code> object as a <code>Browser</code> subclass. . . . .	535
27.10	A student database and browser. . . . .	535
27.11	A trait for food categories. . . . .	536
27.12	A <code>Database</code> class that mixes in the <code>FoodCategories</code> trait. . . . .	536
27.13	A <code>SimpleDatabase</code> object composed solely of mixins. . . . .	536
27.14	A <code>SimpleFoods</code> trait. . . . .	536
27.15	A <code>SimpleRecipes</code> trait with a self type. . . . .	537
27.16	An app that dynamically selects a module implementation. . . . .	538
27.17	Using a singleton type. . . . .	540
28.1	A superclass <code>equals</code> method that calls <code>canEqual</code> . . . . .	555
28.2	A subclass <code>equals</code> method that calls <code>canEqual</code> . . . . .	556
28.3	Hierarchy for binary trees. . . . .	558
28.4	A parameterized type with <code>equals</code> and <code>hashCode</code> . . . . .	562
28.5	Class <code>Rational</code> with <code>equals</code> and <code>hashCode</code> . . . . .	563
29.1	A Scala method that declares a Java <code>throws</code> clause. . . . .	574
30.1	A simple actor. . . . .	585
30.2	An actor that calls <code>receive</code> . . . . .	587

30.3	An actor that calls <code>react</code> . . . . .	591
30.4	An actor's <code>act</code> method that uses <code>loop</code> . . . . .	592
30.5	An actor that uses a helper actor to avoid blocking itself. . . . .	594
30.6	An actor that uses case classes for messages. . . . .	599
30.7	The <code>Simulant</code> trait. . . . .	607
30.8	Adder components. . . . .	614
31.1	An arithmetic expression parser. . . . .	621
31.2	A regular expression parser for Java identifiers. . . . .	623
31.3	Data in JSON format. . . . .	625
31.4	A simple JSON parser. . . . .	625
31.5	A full JSON parser that returns meaningful results. . . . .	630
31.6	The <code>~</code> combinator method. . . . .	638
32.1	A simple Swing application in Scala. . . . .	650
32.2	Component assembly on a panel. . . . .	652
32.3	Implementing a reactive Swing application. . . . .	656
32.4	An implementation of the temperature converter. . . . .	658
33.1	Code for spreadsheet in Figure 33.1. . . . .	663
33.2	The main program for the spreadsheet application. . . . .	664
33.3	A spreadsheet with a <code>renderComponent</code> method. . . . .	665
33.4	First version of the <code>Model</code> class. . . . .	666
33.5	Classes representing formulas. . . . .	668
33.6	A spreadsheet that parses formulas. . . . .	672
33.7	The <code>evaluate</code> method of trait <code>Evaluator</code> . . . . .	675
33.8	A library for arithmetic operations. . . . .	677
33.9	The finished spreadsheet component. . . . .	683