

Kotlin in Action

DMITRY JEMEROV
AND SVETLANA ISAKOVA



MANNING
SHELTER ISLAND

contents

foreword xv
preface xvii
acknowledgments xix
about this book xxi
about the authors xxiv
about the cover illustration xxv

PART 1 INTRODUCING KOTLIN 1

1 *Kotlin: what and why* 3

1.1 A taste of Kotlin 3

1.2 Kotlin's primary traits 4

Target platforms: server-side, Android, anywhere Java runs 4

Statically typed 5 ▪ *Functional and object-oriented* 6

Free and open source 7

1.3 Kotlin applications 7

Kotlin on the server side 8 ▪ *Kotlin on Android* 9

1.4 The philosophy of Kotlin 10

Pragmatic 10 ▪ *Concise* 11 ▪ *Safe* 12 ▪ *Interoperable* 12

1.5 Using the Kotlin tools 13

Compiling Kotlin code 13 ▪ *Plug-in for IntelliJ IDEA and Android*

Studio 14 ▪ *Interactive shell* 15 ▪ *Eclipse plug-in* 15

Online playground 15 ▪ *Java-to-Kotlin converter* 15

1.6 Summary 15

2 **Kotlin basics** 17

- 2.1 Basic elements: functions and variables 18
 - Hello, world!* 18 ▪ *Functions* 18 ▪ *Variables* 20
 - Easier string formatting: string templates* 22
- 2.2 Classes and properties 23
 - Properties* 23 ▪ *Custom accessors* 25 ▪ *Kotlin source code layout: directories and packages* 26
- 2.3 Representing and handling choices: enums and “when” 28
 - Declaring enum classes* 28 ▪ *Using “when” to deal with enum classes* 29 ▪ *Using “when” with arbitrary objects* 30
 - Using “when” without an argument* 31 ▪ *Smart casts: combining type checks and casts* 31 ▪ *Refactoring: replacing “if” with “when”* 33 ▪ *Blocks as branches of “if” and “when”* 34
- 2.4 Iterating over things: “while” and “for” loops 35
 - The “while” loop* 35 ▪ *Iterating over numbers: ranges and progressions* 36 ▪ *Iterating over maps* 37 ▪ *Using “in” to check collection and range membership* 38
- 2.5 Exceptions in Kotlin 39
 - “try”, “catch”, and “finally”* 40 ▪ *“try” as an expression* 41
- 2.6 Summary 42

3 **Defining and calling functions** 44

- 3.1 Creating collections in Kotlin 45
- 3.2 Making functions easier to call 46
 - Named arguments* 47 ▪ *Default parameter values* 48
 - Getting rid of static utility classes: top-level functions and properties* 49
- 3.3 Adding methods to other people’s classes: extension functions and properties 51
 - Imports and extension functions* 53 ▪ *Calling extension functions from Java* 53 ▪ *Utility functions as extensions* 54
 - No overriding for extension functions* 55 ▪ *Extension properties* 56
- 3.4 Working with collections: varargs, infix calls, and library support 57
 - Extending the Java Collections API* 57 ▪ *Varargs: functions that accept an arbitrary number of arguments* 58 ▪ *Working with pairs: infix calls and destructuring declarations* 59

- 3.5 Working with strings and regular expressions 60
 - Splitting strings* 60 ▪ *Regular expressions and triple-quoted strings* 61 ▪ *Multiline triple-quoted strings* 62
- 3.6 Making your code tidy: local functions and extensions 64
- 3.7 Summary 66

4 **Classes, objects, and interfaces** 67

- 4.1 Defining class hierarchies 68
 - Interfaces in Kotlin* 68 ▪ *Open, final, and abstract modifiers: final by default* 70 ▪ *Visibility modifiers: public by default* 73
 - Inner and nested classes: nested by default* 75 ▪ *Sealed classes: defining restricted class hierarchies* 77
- 4.2 Declaring a class with nontrivial constructors or properties 78
 - Initializing classes: primary constructor and initializer blocks* 79
 - Secondary constructors: initializing the superclass in different ways* 81 ▪ *Implementing properties declared in interfaces* 83
 - Accessing a backing field from a getter or setter* 85
 - Changing accessor visibility* 86
- 4.3 Compiler-generated methods: data classes and class delegation 87
 - Universal object methods* 87 ▪ *Data classes: autogenerated implementations of universal methods* 89 ▪ *Class delegation: using the “by” keyword* 91
- 4.4 The “object” keyword: declaring a class and creating an instance, combined 93
 - Object declarations: singletons made easy* 93 ▪ *Companion objects: a place for factory methods and static members* 96
 - Companion objects as regular objects* 98 ▪ *Object expressions: anonymous inner classes rephrased* 100
- 4.5 Summary 101

5 **Programming with lambdas** 103

- 5.1 Lambda expressions and member references 104
 - Introduction to lambdas: blocks of code as function parameters* 104
 - Lambdas and collections* 105 ▪ *Syntax for lambda expressions* 106 ▪ *Accessing variables in scope* 109
 - Member references* 111

- 5.2 Functional APIs for collections 113
 - Essentials: filter and map* 113 ▪ *“all”, “any”, “count”, and “find”: applying a predicate to a collection* 115
 - groupBy: converting a list to a map of groups* 117
 - flatMap and flatten: processing elements in nested collections* 117
 - 5.3 Lazy collection operations: sequences 118
 - Executing sequence operations: intermediate and terminal operations* 120 ▪ *Creating sequences* 122
 - 5.4 Using Java functional interfaces 123
 - Passing a lambda as a parameter to a Java method* 124
 - SAM constructors: explicit conversion of lambdas to functional interfaces* 126
 - 5.5 Lambdas with receivers: “with” and “apply” 128
 - The “with” function* 128 ▪ *The “apply” function* 130
 - 5.6 Summary 131
- 6 The Kotlin type system 133**
- 6.1 Nullability 133
 - Nullable types* 134 ▪ *The meaning of types* 136 ▪ *Safe call operator: “?”* 137 ▪ *Elvis operator: “?:”* 139 ▪ *Safe casts: “as?”* 140 ▪ *Not-null assertions: “!!”* 141 ▪ *The “let” function* 143 ▪ *Late-initialized properties* 145 ▪ *Extensions for nullable types* 146 ▪ *Nullability of type parameters* 148
 - Nullability and Java* 149
 - 6.2 Primitive and other basic types 153
 - Primitive types: Int, Boolean, and more* 153 ▪ *Nullable primitive types: Int?, Boolean?, and more* 154 ▪ *Number conversions* 155
 - “Any” and “Any?”: the root types* 157 ▪ *The Unit type: Kotlin’s “void”* 157 ▪ *The Nothing type: “This function never returns”* 158
 - 6.3 Collections and arrays 159
 - Nullability and collections* 159 ▪ *Read-only and mutable collections* 161 ▪ *Kotlin collections and Java* 163
 - Collections as platform types* 165 ▪ *Arrays of objects and primitive types* 167
 - 6.4 Summary 170

PART 2 EMBRACING KOTLIN 171

7 *Operator overloading and other conventions* 173

- 7.1 Overloading arithmetic operators 174
 - Overloading binary arithmetic operations* 174 ▪ *Overloading compound assignment operators* 177 ▪ *Overloading unary operators* 178
- 7.2 Overloading comparison operators 180
 - Equality operators: “equals”* 180 ▪ *Ordering operators: compareTo* 181
- 7.3 Conventions used for collections and ranges 182
 - Accessing elements by index: “get” and “set”* 182 ▪ *The “in” convention* 184 ▪ *The rangeTo convention* 185 ▪ *The “iterator” convention for the “for” loop* 186
- 7.4 Destructuring declarations and component functions 187
 - Destructuring declarations and loops* 188
- 7.5 Reusing property accessor logic: delegated properties 189
 - Delegated properties: the basics* 189 ▪ *Using delegated properties: lazy initialization and “by lazy()”* 190 ▪ *Implementing delegated properties* 192 ▪ *Delegated-property translation rules* 195
 - Storing property values in a map* 196 ▪ *Delegated properties in frameworks* 197
- 7.6 Summary 199

8 *Higher-order functions: lambdas as parameters and return values* 200

- 8.1 Declaring higher-order functions 201
 - Function types* 201 ▪ *Calling functions passed as arguments* 202 ▪ *Using function types from Java* 204
 - Default and null values for parameters with function types* 205
 - Returning functions from functions* 207 ▪ *Removing duplication through lambdas* 209
- 8.2 Inline functions: removing the overhead of lambdas 211
 - How inlining works* 211 ▪ *Restrictions on inline functions* 213
 - Inlining collection operations* 214 ▪ *Deciding when to declare functions as inline* 215 ▪ *Using inlined lambdas for resource management* 216

- 8.3 Control flow in higher-order functions 217
Return statements in lambdas: return from an enclosing function 217 ▪ *Returning from lambdas: return with a label* 218 ▪ *Anonymous functions: local returns by default* 220
- 8.4 Summary 221

9 Generics 223

- 9.1 Generic type parameters 224
Generic functions and properties 224 ▪ *Declaring generic classes* 226 ▪ *Type parameter constraints* 227
Making type parameters non-null 229
- 9.2 Generics at runtime: erased and reified type parameters 230
Generics at runtime: type checks and casts 230 ▪ *Declaring functions with reified type parameters* 233 ▪ *Replacing class references with reified type parameters* 235 ▪ *Restrictions on reified type parameters* 236
- 9.3 Variance: generics and subtyping 237
Why variance exists: passing an argument to a function 237
Classes, types, and subtypes 238 ▪ *Covariance: preserved subtyping relation* 240 ▪ *Contravariance: reversed subtyping relation* 244 ▪ *Use-site variance: specifying variance for type occurrences* 246 ▪ *Star projection: using * instead of a type argument* 248
- 9.4 Summary 252

10 Annotations and reflection 254

- 10.1 Declaring and applying annotations 255
Applying annotations 255 ▪ *Annotation targets* 256
Using annotations to customize JSON serialization 258
Declaring annotations 260 ▪ *Meta-annotations: controlling how an annotation is processed* 261 ▪ *Classes as annotation parameters* 262 ▪ *Generic classes as annotation parameters* 263
- 10.2 Reflection: introspecting Kotlin objects at runtime 264
The Kotlin reflection API: KClass, KCallable, KFunction, and KProperty 265 ▪ *Implementing object serialization using reflection* 268 ▪ *Customizing serialization with annotations* 270
JSON parsing and object deserialization 273 ▪ *Final deserialization step: callBy() and creating objects using reflection* 277
- 10.3 Summary 281

11	DSL construction	282
11.1	From APIs to DSLs	283
	<i>The concept of domain-specific languages</i>	284
	▪ <i>Internal DSLs</i>	285
	▪ <i>Structure of DSLs</i>	286
	▪ <i>Building HTML with an internal DSL</i>	287
11.2	Building structured APIs: lambdas with receivers in DSLs	288
	<i>Lambdas with receivers and extension function types</i>	288
	<i>Using lambdas with receivers in HTML builders</i>	292
	<i>Kotlin builders: enabling abstraction and reuse</i>	296
11.3	More flexible block nesting with the “invoke” convention	299
	<i>The “invoke” convention: objects callable as functions</i>	299
	<i>The “invoke” convention and functional types</i>	300
	▪ <i>The “invoke” convention in DSLs: declaring dependencies in Gradle</i>	301
11.4	Kotlin DSLs in practice	303
	<i>Chaining infix calls: “should” in test frameworks</i>	303
	▪ <i>Defining extensions on primitive types: handling dates</i>	305
	▪ <i>Member extension functions: internal DSL for SQL</i>	306
	▪ <i>Anko: creating Android UIs dynamically</i>	309
11.5	Summary	310
	<i>appendix A Building Kotlin projects</i>	313
	<i>appendix B Documenting Kotlin code</i>	317
	<i>appendix C The Kotlin ecosystem</i>	320
	<i>index</i>	323