



Wrox Programmer to Programmer™



Programming

# Kotlin® Applications

Building Mobile and Server-Side Applications  
with Kotlin

Brett McLaughlin



# PROGRAMMING KOTLIN® APPLICATIONS

---

INTRODUCTION .....	xxv
CHAPTER 1 Objects All the Way Down.....	1
CHAPTER 2 It's Hard to Break Kotlin.....	25
CHAPTER 3 Kotlin Is Extremely Classy.....	51
CHAPTER 4 Inheritance Matters.....	69
CHAPTER 5 Lists and Sets and Maps, Oh My!.....	101
CHAPTER 6 The Future (in Kotlin) Is Generic.....	129
CHAPTER 7 Flying through Control Structures.....	147
CHAPTER 8 Data Classes.....	183
CHAPTER 9 Enums and Sealed, More Specialty Classes.....	203
CHAPTER 10 Functions and Functions and Functions.....	233
CHAPTER 11 Speaking Idiomatic Kotlin.....	271
CHAPTER 12 Inheritance, One More Time, with Feeling.....	303
CHAPTER 13 Kotlin: The Next Step.....	331
INDEX.....	339



# Programming Kotlin® Applications



# Programming Kotlin® Applications

BUILDING MOBILE AND SERVER-SIDE  
APPLICATIONS WITH KOTLIN

Brett McLaughlin



Copyright © 2021 by John Wiley & Sons, Inc., Indianapolis, Indiana

Published simultaneously in Canada

ISBN: 978-1-119-69618-6

ISBN: 978-1-119-69616-2 (ebk)

ISBN: 978-1-119-69621-6 (ebk)

Manufactured in the United States of America

No part of this publication may be reproduced, stored in a retrieval system or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, scanning or otherwise, except as permitted under Sections 107 or 108 of the 1976 United States Copyright Act, without either the prior written permission of the Publisher, or authorization through payment of the appropriate per-copy fee to the Copyright Clearance Center, 222 Rosewood Drive, Danvers, MA 01923, (978) 750-8400, fax (978) 646-8600. Requests to the Publisher for permission should be addressed to the Permissions Department, John Wiley & Sons, Inc., 111 River Street, Hoboken, NJ 07030, (201) 748-6011, fax (201) 748-6008, or online at [www.wiley.com/go/permissions](http://www.wiley.com/go/permissions).

**Limit of Liability/Disclaimer of Warranty:** The publisher and the author make no representations or warranties with respect to the accuracy or completeness of the contents of this work and specifically disclaim all warranties, including without limitation warranties of fitness for a particular purpose. No warranty may be created or extended by sales or promotional materials. The advice and strategies contained herein may not be suitable for every situation. This work is sold with the understanding that the publisher is not engaged in rendering legal, accounting, or other professional services. If professional assistance is required, the services of a competent professional person should be sought. Neither the publisher nor the author shall be liable for damages arising herefrom. The fact that an organization or Web site is referred to in this work as a citation and/or a potential source of further information does not mean that the author or the publisher endorses the information the organization or Web site may provide or recommendations it may make. Further, readers should be aware that Internet Web sites listed in this work may have changed or disappeared between when this work was written and when it is read.

For general information on our other products and services please contact our Customer Care Department within the United States at (877) 762-2974, outside the United States at (317) 572-3993 or fax (317) 572-4002.

Wiley publishes in a variety of print and electronic formats and by print-on-demand. Some material included with standard print versions of this book may not be included in e-books or in print-on-demand. If this book refers to media such as a CD or DVD that is not included in the version you purchased, you may download this material at [booksupport.wiley.com](http://booksupport.wiley.com). For more information about Wiley products, visit [www.wiley.com](http://www.wiley.com).

**Library of Congress Control Number:** 2020947753

**Trademarks:** Wiley, the Wiley logo, Wrox, the Wrox logo, Programmer to Programmer, and related trade dress are trademarks or registered trademarks of John Wiley & Sons, Inc. and/or its affiliates, in the United States and other countries, and may not be used without written permission. Kotlin is a registered trademark of Kotlin Foundation. All other trademarks are the property of their respective owners. John Wiley & Sons, Inc., is not associated with any product or vendor mentioned in this book.



*for Leigh, as always, my person*



# ABOUT THE AUTHOR

**BRETT MCLAUGHLIN** has been working and writing in the technology space for over 20 years. Today, Brett's focus is squarely on cloud and enterprise computing. He has quickly become a trusted name in helping companies execute a migration to the cloud—and in particular Amazon Web Services—by translating confusing cloud concepts into a clear executive-level vision. He spends his days working with key decision makers who need to understand the cloud as well as leading and building teams of developers and operators who must interact with the ever-changing cloud computing space. He has most recently led large-scale cloud migrations for NASA's Earth Science program and the RockCreek Group's financial platform. Brett is currently the Chief Technology Officer at Volusion, an ecommerce platform provider.



# ABOUT THE TECHNICAL EDITOR

**JASON LEE** is a software developer happily living in the middle of the heartland. He has over 23 years of experience in a variety of languages, writing software running on mobile devices all the way up to big iron. For the past 15+ years, he has worked in the Java/Jakarta EE space, working on application servers, frameworks, and user-facing applications. These days, he spends his time working as a backend engineer, primarily using Kotlin, building systems with frameworks like Quarkus and Spring Boot. He is the author of *Java 9 Programming Blueprints*, a former Java User Group president, an occasional conference speaker, and a blogger. In his spare time, he enjoys spending time with his wife and two sons, reading, playing the bass guitar, and running. He can be found on Twitter at [twitter.com/jasondlee](https://twitter.com/jasondlee), and on his blog at [jasondl.ee](https://jasondl.ee).



# ACKNOWLEDGMENTS

**I USED TO WATCH MOVIES AND STARE** in amazement at the hundreds of names that scrolled by at the end. How could so many people be involved in a single movie?

Then I wrote a book. Now I understand.

Carole Jelen is my agent at Waterside, and she replied to an email and picked up the phone at a time when I really needed someone to help me find my way back into publishing. I'm incredibly grateful.

On the Wiley side, Brad Jones was more patient than he ever should have been. Thanks, Brad! Barath Kumar Rajasekaran handled a million tiny details, and Pete Gaughan and Devon Lewis kept the train on the tracks. Christine O'Connor handled production, and Jason Lee caught the technical mistakes in the text that you wouldn't want to stumble over. Seriously, Jason in particular made this a much better book with his keen eye.

As usual, it's an author's family that pays the highest price. Long days, more than a few weekends and evenings, and a constant support keep us going. My wife, Leigh, is the best, and my kids, Dean, Robbie, and Addie, always make finishing one of these a joy.

Let's do brunch, everyone! Mimosas and breakfast tacos are on me.

—BRETT McLAUGHLIN





# CONTENTS

INTRODUCTION

xxv

## CHAPTER 1: OBJECTS ALL THE WAY DOWN

1

---

Kotlin: A New Programming Language	1
What Is Kotlin?	2
What Does Kotlin Add to Java?	3
Kotlin Is Object-Oriented	3
Interlude: Set Up Your Kotlin Environment	4
Install Kotlin (and an IDE)	4
Install IntelliJ	5
Create Your Kotlin Program	8
Compile and Run Your Kotlin Program	9
Fix Any Errors as They Appear	10
Install Kotlin (and Use the Command Line)	10
Command-Line Kotlin on Windows	10
Command-Line Kotlin on Mac OS X	11
Command-Line Kotlin on UNIX-Based Systems	12
Verify Your Command-Line Installation	12
Creating Useful Objects	13
Pass In Values to an Object Using Its Constructor	13
Print an Object with toString()	14
Terminology Update: Functions and Methods	15
Print an Object (and Do It with Shorthand)	15
Override the toString() Method	16
All Data Is Not a Property Value	17
Initialize an Object and Change a Variable	19
Initialize a Class with a Block	19
Kotlin Auto-Generates Getters and Setters	20
Terminology Update: Getters, Setters, Mutators, Accessors	20
Constants Can't Change (Sort of)	21

## CHAPTER 2: IT'S HARD TO BREAK KOTLIN

25

---

Upgrade Your Kotlin Class Game	25
Name a File According to Its Class	26

---

Organize Your Classes with Packages	27
Put Person in a Package	28
Classes: The Ultimate Type in Kotlin	31
<b>Kotlin Has a Large Number of Types</b>	<b>31</b>
Numbers in Kotlin	31
Letters and Things	32
Truth or Fiction	33
Types Aren't Interchangeable (Part 1)	33
You Must Initialize Your Properties	34
Types Aren't Interchangeable (Part 2)	35
You Can Explicitly Tell Kotlin What Type to Use	36
Try to Anticipate How Types Will Be Used	37
It's Easy to Break Kotlin (Sort of)	37
<b>Overriding Property Accessors and Mutators</b>	<b>37</b>
Custom-Set Properties Can't Be in a Primary Constructor	38
Move Properties Out of Your Primary Constructors	38
Initialize Properties Immediately	39
Try to Avoid Overusing Names	41
Override Mutators for Certain Properties	41
<b>Classes Can Have Custom Behavior</b>	<b>43</b>
Define a Custom Method on Your Class	43
Every Property Must Be Initialized	44
Assign an Uninitialized Property a Dummy Value	45
Tell Kotlin You'll Initialize a Property Later	45
Assign Your Property the Return Value from a Function	46
Sometimes You Don't Need a Property!	47
<b>TYPE SAFETY CHANGES EVERYTHING</b>	<b>49</b>
<b>Writing Code Is Rarely Linear</b>	<b>49</b>
<b>CHAPTER 3: KOTLIN IS EXTREMELY CLASSY</b>	<b>51</b>
<b>Objects, Classes, and Kotlin</b>	<b>51</b>
<b>All Classes Need an equals(x) Method</b>	<b>52</b>
Equals(x) Is Used to Compare Two Objects	52
Override equals(x) to Make It Meaningful	54
Every Object Is a Particular Type	56
A Brief Introduction to Null	58
<b>Every Object Instance Needs a Unique hashCode()</b>	<b>59</b>
All Classes Inherit from Any	59
Always Override hashCode() and equals(x)	61
Default Hash Codes Are Based on Memory Location	63
Use Hash Codes to Come Up with Hash Codes	63

Searching (and Other Things) Depend on Useful and Fast equals(x) and hashCode()	64
Multiple Properties to Differentiate Them in hashCode()	65
Use == over equals(x) for Speed	66
A Quick Speed Check on hashCode()	66
Basic Class Methods Are Really Important	67

---

## CHAPTER 4: INHERITANCE MATTERS 69

---

Good Classes Are Not Always Complex Classes	69
Keep It Simple, Stupid	70
Keep It Flexible, Stupid	71
Classes Can Define Default Values for Properties	73
Constructors Can Accept Default Values	74
Kotlin Expects Arguments in Order	74
Specify Arguments by Name	74
Change the Order of Arguments (If You Need)	75
Secondary Constructors Provide Additional Construction Options	76
Secondary Constructors Come Second	76
Secondary Constructors Can Assign Property Values	77
You Can Assign null to a Property . . . Sometimes	79
null Properties Can Cause Problems	81
Handle Dependent Values with Custom Mutators	82
Set Dependent Values in a Custom Mutator	82
All Property Assignments Use the Property's Mutator	83
Nullable Values Can Be Set to null!	84
Limit Access to Dependent Values	86
When Possible, Calculate Dependent Values	87
You Can Avoid Parentheses with a Read-Only Property	88
Need Specifics? Consider a Subclass	91
Any Is the Base Class for Everything in Kotlin	91
{ . . . } Is Shorthand for Collapsed Code	93
A Class Must Be Open for Subclassing	94
Terminology: Subclass, Inherit, Base Class, and More	95
A Subclass Must Follow Its Superclass's Rules	96
A Subclass Gets Behavior from All of Its Superclasses	96
Your Subclass Should Be Different Than Your Superclass	97
Subclass Constructors Often Add Arguments	97
Don't Make Mutable What Isn't Mutable	98
Sometimes Objects Don't Exactly Map to the Real World	99
Generally, Objects Should Map to the Real World	99

---

<b>CHAPTER 5: LISTS AND SETS AND MAPS, OH MY!</b>	<b>101</b>
<b>Lists Are Just a Collection of Things</b>	<b>101</b>
Kotlin Lists: One Type of Collection	101
Collection Is a Factory for Collection Objects	102
Collection Is Automatically Available to Your Code	104
Mutating a Mutable List	105
Getting Properties from a Mutable List	105
<b>Lists (and Collections) Can Be Typed</b>	<b>106</b>
Give Your Lists Types	107
Iterate over Your Lists	108
Kotlin Tries to Figure Out What You Mean	111
<b>Lists Are Ordered and Can Repeat</b>	<b>111</b>
Order Gives You Ordered Access	112
Lists Can Contain Duplicate Items	112
<b>Sets: Unordered but Unique</b>	<b>113</b>
In Sets, Ordering Is Not Guaranteed	114
When Does Order Matter?	115
Sort Lists (and Sets) on the Fly	115
Sets: No Duplicates, No Matter What	116
Sets “Swallow Up” Duplicates	116
Sets Use equals(x) to Determine Existing Membership	116
Using a Set? Check equals(x)	119
Iterators Aren’t (Always) Mutable	119
<b>Maps: When a Single Value Isn’t Enough</b>	<b>119</b>
Maps Are Created by Factories	120
Use Keys to Find Values	120
How Do You Want Your Value?	121
<b>Filter a Collection by . . . Anything</b>	<b>121</b>
Filter Based on a Certain Criterion	122
Filter Has a Number of Useful Variations	123
<b>Collections: For Primitive and Custom Types</b>	<b>123</b>
Add a Collection to Person	124
Allow Collections to Be Added to Collection Properties	126
Sets and MutableSets Aren’t the Same	127
Collection Properties Are Just Collections	128
<b>CHAPTER 6: THE FUTURE (IN KOTLIN) IS GENERIC</b>	<b>129</b>
<b>Generics Allow Deferring of a Type</b>	<b>129</b>
Collections Are Generic	129
Parameterized Types Are Available Throughout a Class	130
Generic: What Exactly Does It Refer To?	131

---

<b>Generics Try to Infer a Type When Possible</b>	<b>132</b>
Kotlin Looks for Matching Types	132
Kotlin Looks for the Narrowest Type	132
Sometimes Type Inference Is Wrong	133
Don't Assume You Know Object Intent	133
Kotlin Doesn't Tell You the Generic Type	134
Just Tell Kotlin What You Want!	134
<b>Covariance: A Study in Types and Assignment</b>	<b>134</b>
What about Generic Types?	135
Some Languages Take Extra Work to Be Covariant	137
Kotlin Actually Takes Extra Work to Be Covariant, Too	137
Sometimes You Have to Make Explicit What Is Obvious	137
Covariant Types Limit the Input Type as Well as the Output Type	137
Covariance Is Really about Making Inheritance Work the Way You Expect	138
<b>Contravariance: Building Consumers from Generic Types</b>	<b>138</b>
Contravariance: Limiting What Comes Out Rather Than What Comes In	139
Contravariance Works from a Base Class Down to a Subclass	141
Contravariant Classes Can't Return a Generic Type	141
Does Any of This Really Matter?	142
<b>Unsafevariance: Learning The Rules, then Breaking Them</b>	<b>142</b>
<b>Typeprojection Lets You Deal with Base Classes</b>	<b>143</b>
Variance Can Affect Functions, Not Just Classes	143
Type Projection Tells Kotlin to Allow Subclasses as Input for a Base Class	144
Producers Can't Consume and Consumers Can't Produce	145
Variance Can't Solve Every Problem	145
<b>CHAPTER 7: FLYING THROUGH CONTROL STRUCTURES</b>	<b>147</b>
<b>Control Structures Are the Bread and Butter of Programming</b>	<b>147</b>
<b>If and Else: The Great Decision Point</b>	<b>148</b>
!! Ensures Non-Nullable Values	148
Control Structures Affect the Flow of Your Code	149
if and else Follow a Basic Structure	150
Expressions and if Statements	151
Use the Results of an if Statement Directly	152
Kotlin Has No Ternary Operator	153
A Block Evaluates to the Last Statement in That Block	153
if Statements That Are Assigned Must Have else Blocks	154

---

<b>When Is Kotlin's Version of Switch</b>	<b>154</b>
Each Comparison or Condition Is a Code Block	155
Handle Everything Else with an else Block	156
Each Branch Can Support a Range	157
Each Branch Usually Has a Partial Expression	158
Branch Conditions Are Checked Sequentially	159
Branch Conditions Are Just Expressions	159
When Can Be Evaluated as a Statement, Too	160
<b>For Is for Looping</b>	<b>161</b>
For in Kotlin Requires an Iterator	162
You Do Less, Kotlin Does More	163
For Has Requirements for Iteration	163
You Can Grab Indices Instead of Objects with for	164
<b>Use While to Execute until a Condition Is False</b>	<b>167</b>
While Is All about a Boolean Condition	167
A Wrinkle in while: Multiple Operators, One Variable	168
Combine Control Structures for More Interesting Solutions	169
<b>Do . . . While Always Runs Once</b>	<b>170</b>
Every do . . . while Loop Can Be Written as a while Loop	170
If Something Must Happen, Use do . . . while	171
do . . . while Can Be a Performance Consideration	175
<b>Get Out of a Loop Immediately with Break</b>	<b>176</b>
Break Skips What's Left in a Loop	176
You Can Use a Label with break	177
<b>Go to the Next Iteration Immediately with Continue</b>	<b>178</b>
Continue Works with Labels as Well	179
If versus continue: Mostly Style over Substance	179
<b>Return Returns</b>	<b>180</b>
<b>CHAPTER 8: DATA CLASSES</b>	<b>183</b>
<hr/>	
<b>Classes in the Real World Are Varied but Well Explored</b>	<b>183</b>
Many Classes Share Common Characteristics	183
Common Characteristics Result in Common Usage	185
<b>A Data Class Takes the Work Out of a Class Focused on Data</b>	<b>185</b>
Data Classes Handle the Basics of Data for You	185
The Basics of Data Includes hashCode() and equals(x)	186
<b>Destructuring Data through Declarations</b>	<b>188</b>
Grab the Property Values from a Class Instance	188
Destructuring Declarations Aren't Particularly Clever	189

---

Kotlin Is Using componentN() Methods to Make Declarations Work	190
You Can Add componentN() Methods to Any Class	191
If You Can Use a Data Class, You Should	192
You Can “Copy” an Object or Make a Copy Of an Object	192
Using = Doesn’t Actually Make a Copy	192
If You Want a Real Copy, Use copy()	193
Data Classes Require Several Things from You	194
Data Classes Require Parameters and val or var	194
Data Classes Cannot Be Abstract, Open, Sealed, or Inner	195
Data Classes Add Special Behavior to Generated Code	195
You Can Override Compiler-Generated Versions of Many Standard Methods	196
Supertype Class Functions Take Precedence	196
Data Classes Only Generate Code for Constructor Parameters	197
Only Constructor Parameters Are Used in equals()	199
Data Classes Are Best Left Alone	200
<b>CHAPTER 9: ENUMS AND SEALED, MORE SPECIALTY CLASSES</b>	<b>203</b>
<b>Strings Are Terrible as Static Type Representations</b>	<b>203</b>
Strings Are Terrible Type Representations	204
Capitalization Creates Comparison Problems	205
This Problem Occurs All the Time	206
String Constants Can Help . . . Some	206
<b>Companion Objects Are Single Instance</b>	<b>207</b>
Constants Must Be Singular	208
Companion Objects Are Singletons	209
Companion Objects Are Still Objects	210
You Can Use Companion Objects without Their Names	211
Using a Companion Object’s Name Is Optional	211
Using a Companion Object’s Name Is Stylistic	213
Companion Object Names Are Hard	214
You Can Skip the Companion Object Name Altogether	215
<b>Enums Define Constants and Provide Type Safety</b>	<b>216</b>
Enums Classes Provide Type-Safe Values	216
Enums Classes Are Still Classes	218
Enums Give You the Name and Position of Constants	219
Each Constant in an enum Is an Object	219
Each Constant Can Override Class-Level Behavior	220
<b>Sealed Classes Are Type-Safe Class Hierarchies</b>	<b>221</b>

---

Enums and Class Hierarchies Work for Shared Behavior	222
Sealed Classes Address Fixed Options and Non-Shared Behavior	222
Sealed Classes Don't Have Shared Behavior	223
Sealed Classes Have a Fixed Number of Subclasses	224
Subclasses of a Sealed Class Don't Always Define Behavior	225
when Requires All Sealed Subclasses to Be Handled	225
when Expressions Must Be Exhaustive for Sealed Classes	226
else Clauses Usually Don't Work for Sealed Classes	228
else Clauses Hide Unimplemented Subclass Behavior	229
<b>CHAPTER 10: FUNCTIONS AND FUNCTIONS AND FUNCTIONS</b>	<b>233</b>
<b>Revisiting the Syntax of a Function</b>	<b>233</b>
Functions Follow a Basic Formula	233
Function Arguments Also Have a Pattern	235
Default Values in Constructors Are Inherited	237
Default Values in Functions Are Inherited	238
Default Values in Functions Cannot Be Overridden	239
Default Values Can Affect Calling Functions	239
Calling Functions Using Named Arguments Is Flexible	241
Function Arguments Can't Be Null Unless You Say So	241
<b>Functions Follow Flexible Rules</b>	<b>243</b>
Functions Actually Return Unit by Default	243
Functions Can Be Single Expressions	244
Single-Expression Functions Don't Have Curly Braces	245
Single-Expression Functions Don't Use the return Keyword	246
Single-Expression Functions Can Infer a Return Type	246
Type Widening Results in the Widest Type Being Returned	248
Functions Can Take Variable Numbers of Arguments	249
A vararg Argument Can Be Treated Like an Array	251
<b>Functions in Kotlin have Scope</b>	<b>251</b>
Local Functions Are Functions Inside Functions	252
Member Functions Are Defined in a Class	252
Extension Functions Extend Existing Behavior without Inheritance	253
Extend an Existing Closed Class Using Dot Notation	253
this Gives You Access to the Extension Class	255
<b>Function Literals: Lambdas and Anonymous Functions</b>	<b>257</b>
Anonymous Functions Don't Have Names	257
You Can Assign a Function to a Variable	258
Executable Code Makes for an "Executable" Variable	259
Higher-Order Functions Accept Functions as Arguments	260



---

The Result of a Function Is Not a Function	260
Function Notation Focuses on Input and Output	261
You Can Define a Function Inline	263
Lambda Expressions Are Functions with Less Syntax	264
You Can Omit Parameters Altogether	266
Lambda Expressions Use it for Single Parameters	266
It Makes Lambdas Work More Smoothly	267
Lambda Expressions Return the Last Execution Result	267
Trailing Functions as Arguments to Other Functions	268
Lots of Functions, Lots of Room for Problems	268
<b>CHAPTER 11: SPEAKING IDIOMATIC KOTLIN</b>	<b>271</b>
<b>Scope Functions Provide Context to Code</b>	<b>271</b>
<b>Use Let to Provide Immediate Access to an Instance</b>	<b>272</b>
let Gives You it to Access an Instance	273
The Scoped Code Blocks Are Actually Lambdas	274
let and Other Scope functions Are Largely about Convenience	275
You Can Chain Scoped Function Calls	275
An Outer it “Hides” an Inner it	276
Chaining Scope Functions and Nesting Scope Functions Are Not the Same	277
Nesting Scope Functions Requires Care in Naming	277
Chaining Scope Functions Is Simpler and Cleaner	278
Prefer Chaining over Nesting	279
Many Chained Functions Start with a Nested Function	280
You Can Scope Functions to Non-Null Results	280
Accepting null Values Isn’t a Great Idea	282
Scope Functions Give You Null Options	282
Scope Functions Work on Other Functions . . . In Very Particular Ways	284
<b>With Is a Scope Function for Processing an Instance</b>	<b>287</b>
with Uses this as Its Object Reference	287
A this Reference Is Always Available	288
with Returns the Result of the Lambda	289
<b>Run Is a Code Runner and Scope Function</b>	<b>289</b>
Choosing a Scope Function Is a Matter of Style and Preference	290
run Doesn’t Have to Operate on an Object Instance	291
<b>Apply Has a Context Object but No Return Value</b>	<b>292</b>
apply Operates Upon an Instance	292
apply Returns the Context Object, Not the Lambda Result	293
?: Is Kotlin’s Elvis Operator	293
Also Gives You an Instance . . . but Operates on the Instance First	294

---

also Is Just Another Scope Function	295
also Executes before Assignment	296
Scope Functions Summary	298
<b>CHAPTER 12: INHERITANCE, ONE MORE TIME, WITH FEELING</b>	<b>303</b>
<hr/>	
<b>Abstract Classes Require a Later Implementation</b>	<b>303</b>
Abstract Classes Cannot Be Instantiated	304
Abstract Classes Define a Contract with Subclasses	306
Abstract Classes Can Define Concrete Properties and Functions	308
Subclasses Fulfill the Contract Written by an Abstract Class	310
Subclasses Should Vary Behavior	310
The Contract Allows for Uniform Treatment of Subclasses	311
<b>Interfaces Define Behavior but Have No Body</b>	<b>313</b>
Interfaces and Abstract Classes Are Similar	315
Interfaces Cannot Maintain State	316
A Class's State Is the Values of Its Properties	317
An Interface Can Have Fixed Values	317
Interfaces Can Define Function Bodies	318
Interfaces Allow Multiple Forms of Implementation	319
A Class Can Implement Multiple Interfaces	320
Interface Property Names Can Get Confusing	321
Interfaces Can Decorate a Class	321
<b>Delegation Offers Another Option for Extending Behavior</b>	<b>322</b>
Abstract Classes Move from Generic to Specific	322
More Specificity Means More Inheritance	324
Delegating to a Property	326
Delegation Occurs at Instantiation	329
<b>Inheritance Requires Forethought and Afterthought</b>	<b>330</b>
<b>CHAPTER 13: KOTLIN: THE NEXT STEP</b>	<b>331</b>
<hr/>	
<b>Programming Kotlin for Android</b>	<b>331</b>
Kotlin for Android Is Still Just Kotlin	331
Move from Concept to Example	333
<b>Kotlin and Java Are Great Companions</b>	<b>333</b>
Your IDE Is a Key Component	333
Kotlin Is Compiled to Bytecode for the Java Virtual Machine	335
Gradle Gives You Project Build Capabilities	335
<b>When Kotlin Questions Still Exist</b>	<b>335</b>
Use the Internet to Supplement Your Own Needs and Learning Style	336
<b>Now What?</b>	<b>337</b>
<b>INDEX</b>	<b>339</b>

# INTRODUCTION

For decades, the Java programming language has been the dominant force in compiled languages. While there have been plenty of alternatives, it's Java that has remained core to so many applications, from desktop to server-side to mobile. This has become especially true for Android mobile development.

Finally, though, there is a real contender to at least live comfortably beside Java: Kotlin, a modern programming language shepherded by JetBrains ([www.jetbrains.com](http://www.jetbrains.com)). It is not Java, but is completely interoperable with it. Kotlin feels a lot like Java, and will be easy to learn for developers already familiar with the Java language, but offers several nice improvements.

Further, Kotlin is a full-blown programming language. It's not just for mobile applications, or a visual language that focuses on one specific application. Kotlin supports:

- Inheritance, interfaces, implementations, and class hierarchies
- Control and flow structures, both simple and complex
- Lambdas and scope functions
- Rich support for generics while still preserving strong typing
- Idiomatic approaches to development, giving Kotlin a “feel” all its own

You'll also learn that while Kotlin is a new language, it doesn't feel particularly new. That's largely because it builds upon Java, and doesn't try to reinvent wheels. Rather, Kotlin reflects lessons that thousands of programmers coding in Java (and other languages) employ on a daily basis. Kotlin takes many of those lessons and makes them part of the language, enforcing strong typing and a strict compiler that may take some getting used to, but often produces cleaner and safer code.

There's also an emphasis in Kotlin, and therefore in this book, on understanding inheritance. Whether you're using packages from third parties, working with the standard Kotlin libraries, or building your own programs, you need a solid understanding of how classes interrelate, how subclassing works, and how to use abstract classes along with interfaces to define behavior and ensure that behavior is implemented. By the time you're through with this book, you'll be extremely comfortable with classes, objects, and building inheritance trees.

The Kotlin website ([kotlinlang.org](http://kotlinlang.org)) describes Kotlin as “a modern programming language that makes developers happier.” With Kotlin and this book, you'll be happier *and* more productive in your Kotlin programming.

## WHAT DOES THIS BOOK COVER?

This book takes a holistic approach to teaching you the Kotlin programming language, from a beginner to a confident, complete Kotlin developer. By the time you're finished, you'll be able to write Kotlin applications in a variety of contexts, from desktop to server-side to mobile.

## WILL THIS BOOK TEACH ME TO PROGRAM MOBILE APPLICATIONS IN KOTLIN?

---

Yes, but you'll need more than *just* this book to build rich mobile applications in Kotlin. Kotlin is a rich language, and while there are books on all the packages needed to build mobile languages, this is fundamentally a book on learning Kotlin from the ground up. You'll get a handle on how Kotlin deals with generics, inheritance, and lambdas, all critical to mobile programming.

You can then take these concepts and extend them into mobile programming. You can easily add the specifics of Android-related packages to your Kotlin base knowledge, and use those mobile packages far more effectively than if you didn't have the fundamentals down.

If you are anxious to begin your mobile programming journey sooner, consider picking up a book focused on Kotlin mobile programming, and hop back and forth. Read and work through Chapter 1 of this book, and then do the same for the book focused on mobile programming. You'll have to context switch a bit more, but you'll be learning fundamentals alongside specific mobile techniques.

This book covers the following topics:

**Chapter 1: Objects All the Way Down** This chapter takes you from getting Kotlin installed to writing your first Kotlin program. You'll learn about functions from the start, and how to interact with the command line through a not-quite "Hello, World!" application. You'll also immediately begin to see the role of objects and classes in Kotlin, and refine your understanding of what a class is, what an object is, and what an object instance is.

**Chapter 2: It's Hard to Break Kotlin** This chapter delves into one of the distinguishing features of Kotlin: its rigid stance on type safety. You'll learn about Kotlin's types and begin to grasp choosing the right type for the right task. You'll also get familiar with `val` and `var` and how Kotlin allows for change.

**Chapter 3: Kotlin Is Extremely Classy** Like any object-oriented language, much of your work with Kotlin will be writing classes. This chapter digs into classes in Kotlin and looks at the basic building blocks of all Kotlin objects. You'll also override some functions and get deep into some of the most fundamental of Kotlin functions: `equals()` and `hashCode()`.

**Chapter 4: Inheritance Matters** This chapter begins a multichapter journey into Kotlin inheritance. You'll learn about Kotlin's constructors and the relatively unique concept of secondary constructors. You'll also learn more about the `Any` class, understand that inheritance is truly essential for all Kotlin programming, and learn why writing good superclasses is one of the most important skills you can develop in all your programming learning.

**Chapter 5: Lists and Sets and Maps, Oh My!** This chapter moves away (briefly) from classes and inheritance to add Kotlin collections to your arsenal. You'll use these collection classes over and over in your programming, so understanding how a `Set` is different from a `Map`, and how both are different from a `List`, is essential. You'll also dig further into Kotlin mutability and immutability—when data can and cannot change—as well as a variety of ways to iterate over collections of all types.

**Chapter 6: The Future (in Kotlin) Is Generic** Generics are a difficult and nuanced topic in most programming languages. They require a deep understanding of how languages are built. This chapter gets into those depths, and provides you more flexibility in building classes that can be used in a variety of contexts than possible without generics. You'll also learn about covariance, contravariance, and invariance. These might not be the hot topics at the water cooler, but they'll be key to building programs that use generics correctly, and also level up your understanding of inheritance and subclassing.

**Chapter 7: Flying through Control Structures** Control structures are the bread and butter of most programming languages. This chapter breaks down your options, covering `if` and `else`, `when`, `for`, `while`, and `do`. Along the way, you'll focus on controlling the flow of an application or set of applications all while getting a handle on the semantics and mechanics of these structures.

**Chapter 8: Data Classes** This chapter introduces data classes, another very cool Kotlin concept. While not specific to only Kotlin, you'll find that data classes offer you a quick and flexible option for representing data more efficiently than older languages. You'll also really push data classes, going beyond a simple data object and getting into constructors, overriding properties, and both subclassing with and extending from data classes.

**Chapter 9: Enums and Sealed, More Specialty Classes** This chapter introduces enums, a far superior approach to `String` constants. You'll learn why using `Strings` for constant values is a really bad idea, and how enums give you greater flexibility and type safety, as well as making your code easier to write. From enums, you'll move into sealed classes, a particularly cool feature of Kotlin that lets you turbo-charge the concept of enums even further. You'll also dig into companion objects and factories, all of which contribute to a robust type-safe approach to programming where previously only `String` types were used.

**Chapter 10: Functions and Functions and Functions** It may seem odd to have a chapter this late in the book that purports to focus on functions. However, as with most fundamentals in any discipline, you'll have to revisit the basics over and over again, shoring up weaknesses and adding nuance. This chapter does just that with functions. You'll dig more deeply into just how arguments really work, and how many options Kotlin provides to you in working with data going into and out of your functions.

**Chapter 11: Speaking Idiomatic Kotlin** Kotlin, like all programming languages, has certain patterns of usage that seasoned programmers revert to time and time again. This chapter discusses these and some of the idioms of Kotlin. You'll get a jump start on writing Kotlin that looks like Kotlin is “supposed to” all while understanding how you have a tremendous amount of flexibility in choosing how to make your Kotlin programs feel like “you.”

**Chapter 12: Inheritance, One More Time, with Feeling** Yes, it really is another chapter on inheritance! This chapter takes what you’ve already learned about abstract classes and superclasses and adds interfaces and implementations into the mix. You’ll also learn about the delegation pattern, a common Kotlin pattern that helps you take inheritance even further with greater flexibility than inheritance alone provides.

**Chapter 13: Kotlin: The Next Step** No book can teach you everything you need to know, and this book is certainly no exception. There are some well-established places to look for next steps in your Kotlin programming journey, though, and this chapter gives you a number of jumping-off points to continue learning about specific areas of Kotlin.

## Reader Support for This Book Companion Download Files

As you work through the examples in this book, the project files you need are available for download from [www.wiley.com/go/programmingkotlinapplications](http://www.wiley.com/go/programmingkotlinapplications).

## How to Contact the Publisher

If you believe you’ve found a mistake in this book, please bring it to our attention. At John Wiley & Sons, we understand how important it is to provide our customers with accurate content, but even with our best efforts an error may occur.

In order to submit your possible errata, please email it to our Customer Service Team at [wileysupport@wiley.com](mailto:wileysupport@wiley.com) with the subject line “Possible Book Errata Submission.”

## How to Contact the Author

We appreciate your input and questions about this book! Email me at [brett@brettdmclaughlin.com](mailto:brett@brettdmclaughlin.com), or DM me on Twitter at [@bdmclaughlin](https://twitter.com/bdmclaughlin).