Jetpack Compose Essentials

Jetpack Compose Essentials

ISBN-13: 978-1-951442-38-5

© 2022 Neil Smyth / Payload Media, Inc. All Rights Reserved.

This book is provided for personal use only. Unauthorized use, reproduction and/or distribution strictly prohibited. All rights reserved.

The content of this book is provided for informational purposes only. Neither the publisher nor the author offers any warranties or representation, express or implied, with regard to the accuracy of information contained in this book, nor do they accept any liability for any loss or damage arising from any errors or omissions.

This book contains trademarked terms that are used solely for editorial purposes and to the benefit of the respective trademark owner. The terms used within this book are not intended as infringement of any trademarks.

Rev: 1.0

Contents

Table of Contents

1. Start Here	1
1.1 For Kotlin programmers	1
1.2 For new Kotlin programmers	1
1.3 Downloading the code samples.	1
1 4 Feedback	2
1.5 Frrata	2
 Setting up an Android Studio Development Environment 	2
2. Setting up un Android Studio Development Environment	
2.1 System requirements.	3
2.2 Downloading the Android Studio package	3
2.3 Installing Android Studio	4
2.3.1 Installation on Windows	4
2.3.2 Installation on macOS	4
2.3.3 Installation on Linux	5
2.4 The Android Studio setup wizard	5
2.5 Installing additional Android SDK packages	6
2.6 Making the Android SDK tools command-line accessible	9
2.6.1 Windows 8.1	9
2.6.2 Windows 10	10
2.6.3 Windows 11	10
2.6.4 Linux	10
2.6.5 macOS	10
2.7 Android Studio memory management	11
2.8 Updating Android Studio and the SDK	11
2.9 Summary	11
3. A Compose Project Overview	
3.1 About the project	13
3.2 Creating the project	14
3.3 Creating an activity	14
3.4 Defining the project and SDK settings	15
3.5 Draviaving the avample project	15
3.6 Paviawing the main activity	10
3.7 Draviou undatos	10
3.7 Fleview updates	
4 An Example Compass Designt	22
4. An Example Compose Project	
4.1 Getting started	23
4.2 Removing the template Code	23
4.3 The Composable hierarchy	24
4.4 Adding the DemoText composable	24
4.5 Previewing the DemoText composable	26
4.6 Adding the DemoSlider composable	26
4.7 Adding the DemoScreen composable	27

4.8 Previewing the DemoScreen composable	
4.9 Testing in interactive mode	
4.10 Completing the project	
4.11 Summary	
5. Creating an Android Virtual Device (AVD) in Android Studio	
5.1 About Android Virtual Devices	
5.2 Starting the emulator	
5.3 Running the application in the AVD	
5.4 Running on multiple devices	
5.5 Stopping a running application	
5.6 Supporting dark theme	
5.7 Running the emulator in a separate window	
5.8 Enabling the device frame	
5.9 AVD command-line creation	
5.10 Android Virtual Device configuration files	
5.11 Moving and renaming an Android Virtual Device	
5.12 Summary	
6. Using and Configuring the Android Studio AVD Emulator	
6.1 The emulator environment	45
6.2 The emulator toolbar options	45
6.3 Working in zoom mode	47
6.4 Resizing the emulator window	47 47
6.5 Extended control ontions	47
6.5.1 Location	48
6.5.2 Displays	48
6.5.3 Cellular	48
6.5.4 Battery	48
6.5.5 Camera	48
6.5.6 Phone	48
6.5.7 Directional nad	48
6.5.8 Microphone	48
6 5 9 Fingerprint	48
6.5.10 Virtual sensors	49
6.5.11 Snapshots	49
6.5.12 Record and playback	49
6.5.13 Google Play	49
6 5 14 Settings	49
6 5 1 5 Help	49
6.6 Working with snapshots	49
6.7 Configuring fingerprint emulation	50
6.8 The emulator in tool window mode	51
6.9 Summary	
7. A Tour of the Android Studio User Interface	53
7.1 The Welcome screen	53
7.2 The main window	54
7.3 The tool windows	55
7.4 Android Studio keyboard shortcuts	58
7.5 Switcher and recent files navigation	59
, le officier und recent mes nuvigation	

7.6 Changing the Android Studio theme	60
7.7 Summary	60
8. Testing Android Studio Apps on a Physical Android Device	61
8.1 An overview of the Android Debug Bridge (ADB)	
8.2 Enabling USB debugging ADB on Android devices	
8.2.1 macOS ADB configuration	
8.2.2 Windows ADB configuration	
8.2.3 Linux adb configuration	
8.3 Resolving USB connection issues	
8.4 Enabling wireless debugging on Android devices	
8.5 Testing the adb connection	
8.6 Summary	
9. The Basics of the Android Studio Code Editor	
9.1 The Android Studio editor	69
9.2 Code mode	
9.3 Splitting the editor window	
9.4 Code completion	
9.5 Statement completion	
9.6 Parameter information	
9.7 Parameter name hints	
9.8 Code generation	74
9.9 Code folding	75
9.10 Quick documentation lookup	77
9.11 Code reformatting	77
9.12 Finding sample code	
9.13 Live templates	
9.14 Summary	79
10. An Overview of the Android Architecture	
10.1 The Android software stack	
10.2 The Linux kernel	82
10.3 Android runtime – ART	
10.4 Android libraries	
10.4.1 C/C++ libraries	
10.5 Application framework	
10.6 Applications	
10.7 Summary	
11. An Introduction to Kotlin	
11.1 What is Kotlin?	
11.2 Kotlin and Java	
11.3 Converting from Java to Kotlin	
11.4 Kotlin and Android Studio	
11.5 Experimenting with Kotlin	
11.6 Semi-colons in Kotlin	
11.7 Summary	
12. Kotlin Data Types, Variables and Nullability	
12.1 Kotlin data types	

12.1.1 Integer data types	
12.1.2 Floating point data types	
12.1.3 Boolean data type	
12.1.4 Character data type	
12.1.5 String data type	
12.1.6 Escape sequences	
12.2 Mutable variables	
12.3 Immutable variables	
12.4 Declaring mutable and immutable variables	
12.5 Data types are objects	
12.6 Type annotations and type inference	
12.7 Nullable type	
12.8 The safe call operator	
12.9 Not-null assertion	95
12.10 Nullable types and the let function	
12.11 Late initialization (lateinit)	96
12.12 The Elvis operator	97
12.12 The Error operator inner the checking	97
12.15 Type custing and type encerting	98
13. Kotlin Operators and Expressions	
13.1 Expression syntax in Kotlin	
13.2 The Basic assignment operator	
13.3 Kotlin arithmetic operators	
13.4 Augmented assignment operators	
13.5 Increment and decrement operators	
13.6 Equality operators	
13.7 Boolean logical operators	
13.8 Range operator	
13.9 Bitwise operators	
13.9.1 Bitwise Inversion	
13.9.2 Bitwise AND	
13.9.3 Bitwise OR	
13.9.4 Bitwise XOR	
13.9.5 Bitwise left shift	
13.9.6 Bitwise right shift	
13.10 Summary	
14. Kotlin Control flow	
14.1 Looming control flows	107
14.1 Looping control now	
14.1.1 The Kotin <i>jor-in</i> Statement	
14.1.2 The write loop	
14.1.5 Ine <i>ao while</i> loop	
14.1.4 Breaking from Loops	
14.1.5 Ine continue statement	
14.1.6 Break and continue labels	
14.2 Conditional control flow	
14.2.1 Using the ij expressions	
14.2.2 Using $if \dots else \dots$ expressions	
14.2.3 Using <i>if else if</i> Expressions	

14.2.4 Using the when statement	
14.3 Summary	
15. An Overview of Kotlin Functions and Lambdas	
15.1 What is a function?	
15.2 How to declare a Kotlin function	
15.3 Calling a Kotlin function	
15.4 Single expression functions	
15.5 Local functions	
15.6 Handling return values	
15.7 Declaring default function parameters	
15.8 Variable number of function parameters	
15.9 Lambda expressions	
15.10 Higher-order functions	
15.11 Summary	
16. The Basics of Object-Oriented Programming in Kotlin	
16.1 What is an object?	
16.2 What is a class?	
16.3 Declaring a Kotlin class	
16.4 Adding properties to a class	
16.5 Defining methods	
16.6 Declaring and initializing a class instance	
16.7 Primary and secondary constructors	
16.8 Initializer blocks	
16.9 Calling methods and accessing properties	
16.10 Custom accessors	
16.11 Nested and inner classes	
16.12 Companion objects	
16.13 Summary	
17. An Introduction to Kotlin Inheritance and Subclassing	
17.1 Inheritance, classes, and subclasses	
17.2 Subclassing syntax	
17.3 A Kotlin inheritance example	
17.4 Extending the functionality of a subclass	
17.5 Overriding inherited methods	
17.6 Adding a custom secondary constructor	
17.7 Using the SavingsAccount class	
17.8 Summary	
18. An Overview of Compose	
18.1 Development before Compose	
18.2 Compose declarative syntax	
18.3 Compose is data-driven	
18.4 Summary	
19. Composable Functions Overview	
19.1 What is a composable function?	141
19.2 Stateful vs. stateless composables	141
19.3 Composable function syntax	

19.4 Foundation and Material composables	
19.5 Summary	
20. An Overview of Compose State and Recomposition	
20.1 The basics of state	147
20.2 Introducing recomposition	
20.3 Creating the StateExample project	
20.4 Declaring state in a composable	
20.5 Unidirectional data flow	
20.6 State hoisting	
20.7 Saving state through configuration changes	
20.8 Summary	
21. An Introduction to Composition Local	
21.1 Understanding CompositionLocal	
21.2 Using CompositionLocal	
21.3 Creating the CompLocalDemo project	
21.4 Designing the layout	
21.5 Adding the CompositionLocal state	
21.6 Accessing the CompositionLocal state	
21.7 Testing the design	
21.8 Summary	
22. An Overview of Compose Slot APIs	
22.1 Understanding slot APIs	165
22.2 Declaring a slot API	
22.3 Calling slot API composables	
22.4 Summary	
23. A Compose Slot API Tutorial	
23.1 About the project	169
23.2 Creating the SlotApiDemo project	169
23.3 Preparing the Main Activity class file	
23.4 Creating the MainScreen composable	
23.5 Adding the ScreenContent composable	
23.6 Creating the Checkbox composable	
23.7 Implementing the ScreenContent slot API	
23.8 Adding an Image drawable resource	
23.9 Writing the TitleImage composable	
23.10 Completing the MainScreen composable	
23.11 Previewing the project	
23.12 Summary	
24. Using Modifiers in Compose	
24.1 An overview of modifiers	
24.2 Creating the ModifierDemo project	
24.3 Creating a modifier	
24.4 Modifier ordering	
24.5 Adding modifier support to a composable	
24.6 Common built-in modifiers	
24.7 Combining modifiers	

24.8 Summary	
25. Composing Layouts with Row and Column	
25.1 Creating the RowColDemo project	
25.2 Row composable	
25.3 Column composable	
25.4 Combining Row and Column composables	
25.5 Lavout alignment	194
25.6 Layout arrangement positioning	196
25.7 Layout arrangement spacing	198
25.8 Row and Column scope modifiers	199
25.9 Scope modifier weights	203
25.5 Scope mounter weights	203
26 Box Lavouts in Compose	205
26.1 An introduction to the Box composable	205
26.2 Creating the Boyl avoit project	205
26.2 Adding the TextCall composable	205
26.4 Adding a Powlewout	205
26.5 Pox alignment	208
20.5 DOX alignment.	
26.6 BoxScope modifiers	
26.7 Using the clip() modifier	
26.8 Summary	
27. Custom Layout Modifiers	
27.1 Compose layout basics	
27.2 Custom layouts	
27.3 Creating the LayoutModifier project	
27.4 Adding the ColorBox composable	
27.5 Creating a custom layout modifier	
27.6 Understanding default position	
27.7 Completing the layout modifier	
27.8 Using a custom modifier	
27.9 Working with alignment lines	
27.10 Working with baselines	
27.11 Summary	
28. Building Custom Layouts	
28.1 An overview of custom layouts	
28.2 Custom layout syntax	
28.3 Using a custom layout	
28.4 Creating the CustomLayout project	
28.5 Creating the CascadeLayout composable	
28.6 Using the CascadeLayout composable	
28.7 Summary	
29. A Guide to ConstraintLayout in Compose	
29.1 An introduction to Constraint avout	
29.2 How ConstraintLavout works.	2.2.7
29.2.1 Constraints	227
29.2.2 Margins	227

29.2.4 Constraint bias 229 29.2.6 Chain styles 230 29.2.6 Chain styles 230 29.3 Configuring dimensions 231 29.4 Guideline helper 231 29.5 Barrier helper 232 29.6 Summary 233 30. Working with ConstraintLayout in Compose 235 30.1 Calling ConstraintLayout 235 30.2 Generating references 235 30.3 Assigning a reference to a composable 235 30.5 Creating the ConstraintJayout project 236 30.6 Adding the ConstraintJayout project 236 30.7 Adding a custom button composable 237 30.8 Basic constraints 238 30.9 Opposing constraints 239 30.10 Constraint bias 241 30.11 Constraint margins 241 30.12 The importance of opposing constraints and bias 242 30.13 Creating with barriers 246 30.14 Working with guidelines 246 30.15 Working with neasurements 252 31.1 Mux vs Min. IntrinsicSize in Compose 253 31.2 Max vs Min. IntrinsicSize nop roject. 254 31.		29.2.3 Opposing constraints	228	
29.2.5 Chains 230 29.2.6 Chain styles 230 29.3 Configuring dimensions 231 29.4 Guideline helper 231 29.5 Barrier helper 232 29.6 Summary 233 30. Working with ConstraintLayout in Compose 235 30.1 Calling ConstraintLayout 235 30.1 Calling ConstraintLayout 235 30.3 Assigning a references 235 30.4 Adding constraints 236 30.5 Creating the ConstraintLayout project 236 30.6 Adding to constraintLayout project 236 30.6 Adding a custom button composable 237 30.7 Adding a custom button composable 237 30.8 Basic constraints 238 30.9 Opposing constraints 239 30.10 Constraint bas 240 30.11 Constraint margins 241 30.12 Creating chains 242 30.13 Creating chains 242 30.14 Working with burdelines 243 30.15 Creating chains 241 30.15 Creating chains 242 30.16 Decoupling constraints with constraint sets 250		29.2.4 Constraint bias	229	
29.2.6 Chain styles 230 29.3 Configuring dimensions 231 29.4 Guideline helper 231 29.5 Barrier helper 232 29.6 Summary 233 30. Working with ConstraintLayout in Compose 235 30.1 Calling ConstraintLayout 235 30.2 Generating references 235 30.3 Assigning a reference to a composable 235 30.4 Adding the ConstraintLayout project 236 30.5 Creating the ConstraintLayout project 236 30.6 Adding the ConstraintLayout bibrary 237 30.7 Adding a custom button composable 237 30.7 Adding a custom button composable 237 30.8 Opposing constraints 239 30.10 Constraint bias 240 30.11 Constraint margins 241 30.12 Creating chains 242 30.13 Creating this constraints and bias 242 30.14 Working with barriers 247 30.15 Decoupling constraints with constraint sets 252 31. Working with IntrinsicSize in Compose 253 31.1 About the example project 254 31.3 About the example project 254		29.2.5 Chains	230	
29.3 Configuring dimensions 231 29.4 Guideline helper 231 29.5 Barrier helper 232 29.6 Summary 233 30. Working with ConstraintLayout in Compose 235 30.1 Calling ConstraintLayout, 235 30.3 Assigning a references 235 30.4 Adding constraints 236 30.5 Creating the ConstraintLayout project 236 30.6 Adding to constraint. 237 30.7 Adding a custom button composable 237 30.7 Adding a custom button composable 237 30.8 assigning a custom button composable 237 30.9 Opposing constraints 238 30.9 Opposing constraints 238 30.9 Opposing constraints 240 30.1 Constraint margins 241 30.1 Constraint margins 241 30.1 Constraint with constraint sets 246 30.1 Working with barriers 247 30.1 Constraint sets 250 30.1 Fourbard with barriers 251 31.2 Max. vs Min. Intrinsic Size in Compose 253 31.3 About the example project 253 31.4 Max. vs Min. Intrinsic		29.2.6 Chain styles	230	
29.4 Guideline helper. 231 29.5 Barrier helper. 232 29.6 Summary 233 30. Working with ConstraintLayout in Compose 235 30.1 Calling ConstraintLayout, 235 30.1 Calling constraintLayout project. 235 30.4 Adding constraintLayout project. 236 30.5 Creating the ConstraintLayout library 237 30.7 Adding a custom button composable 237 30.8 asia constraints. 238 30.9 Opposing constraints. 238 30.9 Opposing constraints. 239 30.10 Constraint bias. 240 30.11 Constraint margins 241 30.12 The importance of opposing constraints and bias 242 30.13 Creating chains. 246 30.14 Decorphing constraints with constraint sets. 250 30.14 Working with uidelines. 246 30.15 Working with barriers. 246 30.15 Working with IntrinsicSize in Compose 253 31.1 Mutrinsic Size measurements 253 31.3 About the example project. 254 31.4 Creating the IntrinsicSize Dem project. 255 31.5 Creating the project.		29.3 Configuring dimensions	231	
29.5 Barrier helper. 232 29.6 Summary 233 30. Working with ConstraintLayout in Compose 235 30.1 Calling ConstraintLayout. 235 30.2 Generating references to a composable 235 30.3 Assigning a reference to a composable 235 30.4 Adding constraintLayout project 236 30.5 Creating the ConstraintLayout project 236 30.6 Adding a custom button composable 237 30.8 Basic constraints. 238 30.9 Opposing constraints. 238 30.9 Opposing constraints. 240 30.11 Constraint margins 241 30.12 The importance of opposing constraints and bias 242 30.14 Working with guidelines 247 30.15 Working with puidelines 246 30.17 Summary 252 31. Untrinsic Size in Compose 253 31.1 Intrinsic Size in Compose 253 31.2 Max, vs Min, Intrinsic Size measurements 253 31.3 About the example project 254 31.4 Creating the IntrinsicSizeDemo project 254 31.4 Creating the IntrinsicSize Max measurements 255 31.6 Adding the Tex		29.4 Guideline helper	231	
29.6 Summary		29.5 Barrier helper	232	
30. Working with ConstraintLayout in Compose 235 30.1 Calling ConstraintLayout 235 30.2 Generating references 235 30.3 Assigning a reference to a composable 236 30.4 Adding constraints 236 30.5 Creating the ConstraintLayout project 236 30.6 Adding the ConstraintLayout project 237 30.7 Adding a custom button composable 237 30.8 Basic constraints 238 30.9 Opposing constraints 239 30.10 Constraint bias 240 30.11 Constraint margins 241 30.12 Chevicking with guidelines 246 30.14 Working with guidelines 246 30.15 Opcoking onstraints with constraint sets 247 30.16 Decoupling constraints with constraint sets 250 30.17 Summary 252 31.1 Intrinsic measurements 253 31.2 Max. vs Min. Intrinsic Size measurements 253 31.3 About the example project 254 31.4 Creating the IntrinsicSizeDemo project 254 31.4 Creating the transicSizeDemo project 255 31.5 Adding the top-level Column 256 31.7 Addin		29.6 Summary	233	
30.1 Calling ConstraintLayout 235 30.2 Generating references 235 30.3 Assigning a reference to a composable 235 30.4 Adding constraints 236 30.5 Creating the ConstraintLayout project 236 30.6 Adding a custom button composable 237 30.7 Adding a custom button composable 237 30.8 Basic constraints 238 30.9 Opposing constraints 239 30.10 Constraint bias 240 30.11 Constraint bias 241 30.12 The importance of opposing constraints and bias 242 30.13 Constraint bias 244 30.14 Working with guidelines 246 30.15 Working with guidelines 246 30.17 Summary 252 31.1 Intrinsic measurements 253 31.2 Max. vs Min. IntrinsicSize in Compose 253 31.3 About the example project. 254 31.4 Creating the IntrinsicSizeDemo project. 255 31.5 Creating the IntrinsicSizeDemo project. 255 31.6 Adding the Text and Box components. 256 31.7 Adduting the top-level Column 256 31.8 Testing the project <td< td=""><td>30.</td><td>Working with ConstraintLayout in Compose</td><td></td><td>235</td></td<>	30.	Working with ConstraintLayout in Compose		235
30.2 Generating references 235 30.3 Assigning a reference to a composable 235 30.4 Adding constraints 236 30.5 Creating the ConstraintLayout project 236 30.6 Adding the ConstraintLayout library 237 30.7 Adding a custom button composable 237 30.7 Adding a custom button composable 238 30.9 Opposing constraints 238 30.9 Opposing constraints 239 30.10 Constraint margins 241 30.11 Constraint margins 241 30.12 The importance of opposing constraints and bias 242 30.13 Creating chains 246 30.14 Working with guidelines 246 30.15 Working with barriers 247 30.16 Decoupling constraints with constraint sets 250 30.17 Summary 252 31.1 Mutrinsic measurements 253 31.2 Max. vs Min. IntrinsicSize in Compose 253 31.3 About the example project 255 31.5 Creating the furthinsicSizeDemo project 255 31.5 Creating the furthinsicSizeDemo project 256 31.7 Adding the top-level Column 256 31.8 Adding the		30.1 Calling ConstraintLayout	235	
30.3 Assigning a reference to a composable 235 30.4 Adding constraints 236 30.5 Creating the ConstraintLayout project 237 30.6 Adding the ConstraintLayout library 237 30.7 Adding a custom button composable 237 30.8 Basic constraints 238 30.9 Opposing constraints 239 30.10 Constraint bias 240 30.11 Constraint margins 241 30.13 Creating chains 245 30.14 Working with guidelines 246 30.15 Working with guidelines 247 30.16 Decoupling constraints sets 250 30.17 Summary 252 31. Working with IntrinsicSize in Compose 253 31.1 Intrinsic measurements 253 31.2 Max, vs Min. IntrinsicSize Demo project 255 31.5 Creating the ustom text field 255 31.6 Adding the Text and Box components 256 31.7 Adding the top-level Column 256 31.7 Adding the top-level Column 256 31.1 MarinsicSize.Max measurements 257 31.9 Applying IntrinsicSize.Max measurements 256 31.7 Adding the top-level Column		30.2 Generating references		
30.4 Adding constraints. 236 30.5 Creating the ConstraintLayout project. 236 30.6 Adding the ConstraintLayout library. 237 30.7 Adding a custom button composable. 237 30.8 Basic constraints. 238 30.9 Opposing constraints. 239 30.10 Constraint bias. 240 30.11 Constraint margins 241 30.12 The importance of opposing constraints and bias. 242 30.13 Creating chains. 246 30.14 Working with burriers. 246 30.15 Working with barriers. 247 30.16 Decoupling constraints with constraint sets. 250 30.17 Summary. 252 31.1 Intrinsic measurements 253 31.2 Max. vs Min. Intrinsic Size measurements. 253 31.3 About the example project. 254 31.4 Creating the turb micsize/zeDemo project. 255 31.6 Adding the Text and Box components. 256 31.7 Adding the top-level Column. 256 31.8 Testing the project. 257 31.9 Applying IntrinsicSize.Max measurements 257 31.10 Applying IntrinsicSize.Max measurements 257 <t< td=""><td></td><td>30.3 Assigning a reference to a composable</td><td></td><td></td></t<>		30.3 Assigning a reference to a composable		
30.5 Creating the ConstraintLayout project 236 30.6 Adding the ConstraintLayout library 237 30.7 Adding a custom button composable 237 30.7 Adding a custom button composable 237 30.8 Basic constraints 238 30.9 Opposing constraints 239 30.10 Constraint bias 240 30.11 Constraint margins 241 30.12 The importance of opposing constraints and bias 242 30.13 Creating chains 245 30.14 Working with pariers 246 30.15 Working with barriers 247 30.16 Decoupling constraints with constraint sets 250 30.17 Summary 252 31.1 Intrinsic Tisc measurements 253 31.2 Max, vs Min. Intrinsic Size in Compose 253 31.3 About the example project 254 31.4 Creating the custom text field 255 31.5 Creating the custom text field 255 31.6 Adding the Text and Box components 256 31.7 Adding the top-level Column 256 31.7 Adding the top-level Column 256 31.7 Adding the Text and Box c		30.4 Adding constraints	236	
30.6 Adding the ConstraintLayout library 237 30.7 Adding a custom button composable 237 30.8 Basic constraints 238 30.9 Opposing constraints 239 30.10 Constraint bias 240 30.11 Constraint margins 241 30.12 The importance of opposing constraints and bias 242 30.13 Creating chains 245 30.14 Working with guidelines 246 30.15 Working with barriers 247 30.16 Decoupling constraints with constraint sets 250 30.17 Summary 252 31.1 Intrinsic measurements 253 31.2 Max. vs Min. Intrinsic Size measurements 253 31.3 About the example project 254 31.4 Creating the IntrinsicSizeDemo project 255 31.5 Creating the custom text field 255 31.6 Adding the Text and Box components 256 31.7 Adding the top-level Column 256 31.8 About the sample project 257 31.9 Applying IntrinsicSize Max measurements 257 31.4 Kreating the project 257 31.5 Creating the project 256 31.7 Adding the Top-level Column <td></td> <td>30.5 Creating the ConstraintLayout project</td> <td>236</td> <td></td>		30.5 Creating the ConstraintLayout project	236	
30.7 Adding a custom button composable 237 30.8 Basic constraints 238 30.9 Opposing constraints 239 30.10 Constraint bias 240 30.11 Constraint margins 241 30.12 The importance of opposing constraints and bias 242 30.13 Creating chains 245 30.14 Working with guidelines 246 30.15 Working with barriers 247 30.16 Decoupling constraints with constraint sets 250 30.17 Summary 252 31. Working with IntrinsicSize in Compose 253 31.1 Intrinsic measurements 253 31.2 Max. vs Min. IntrinsicSize measurements 253 31.3 About the example project 254 31.4 Creating the IntrinsicSizeDemo project 255 31.5 Creating the custom text field 255 31.6 Adding the top-level Column 256 31.7 Adding the top-level Column 256 31.9 Applying IntrinsicSize.Max measurements 257 31.1 May and the set and Box components 257 31.4 Adding the top-level Column 256 31.5 Creating the project 257 31.9 Applying IntrinsicSi		30.6 Adding the ConstraintLayout library	237	
30.8 Basic constraints 23 30.9 Opposing constraints 239 30.10 Constraint bias 240 30.11 Constraint margins 241 30.12 The importance of opposing constraints and bias 242 30.13 Creating chains 242 30.14 Working with guidelines 246 30.15 Working with barriers 247 30.16 Decoupling constraints with constraint sets 250 30.17 Summary 252 31. Working with IntrinsicSize in Compose 253 31.1 Intrinsic measurements 253 31.2 Max. vs Min. Intrinsic Size Demo project 254 31.4 Creating the IntrinsicSizeDemo project 255 31.5 Creating the custom text field 255 31.6 Adding the Text and Box components 256 31.7 Adding the top-level Column 256 31.8 Testing the project 257 31.9 Applying IntrinsicSize.Max measurements 257 31.1 Isummary 258 32. An Overview of Lists and Grids in Compose 258 32. An Overview of Lists and Grids in Compose 259 32.1 Standard vs. lazy lists 259 32.2 Working with Column and		30.7 Adding a custom button composable	237	
30.9 Opposing constraints 239 30.10 Constraint bias 240 30.11 Constraint margins 241 30.12 The importance of opposing constraints and bias 242 30.13 Creating chains 245 30.14 Working with guidelines 246 30.15 Working with barriers 247 30.16 Decoupling constraints with constraint sets 250 30.17 Summary 252 31.1 Intrinsic Size in Compose 253 31.2 Max, vs Min. Intrinsic Size measurements 253 31.3 About the example project 254 31.4 Creating the IntrinsicSizeDemo project 255 31.5 Creating the Custom text field 255 31.6 Adding the Text and Box components 256 31.7 Adding the top-level Column 256 31.8 Testing the project 257 31.9 Applying IntrinsicSize.Max measurements 257 31.1 Numary 258 32.1 Standard vs. lazy lists 259 32.2 Morking with Column and Row lists 259 32.3 Creating lazy lists 259 32.4 Ereating and the scrolling with ScrollState 261 32.5 Programmatic scrolling <		30.8 Basic constraints	238	
30.10 Constraint bias. 240 30.11 Constraint margins 241 30.12 The importance of opposing constraints and bias. 242 30.13 Creating chains. 245 30.14 Working with guidelines. 246 30.15 Working with barriers 247 30.16 Decoupling constraints with constraint sets. 250 30.17 Summary. 252 31. Working with IntrinsicSize in Compose. 253 31.2 Max. vs Min. Intrinsic Size measurements 253 31.3 About the example project. 254 31.4 Creating the IntrinsicSizeDemo project. 255 31.5 Creating the custom text field. 255 31.6 Adding the Text and Box components. 256 31.7 Adding the top-level Column 256 31.7 Adding the top-level Column 256 31.9 Applying IntrinsicSize.Max measurements 257 31.10 Applying IntrinsicSize.Max measurements 257 31.11 Summary. 258 32.1 Standard vs. lazy lists 259 32.2 Working with Column and Row lists 259 32.3 Creating lazy lists 259 32.4 Enabling scrolling with ScrollState 260		30.9 Opposing constraints	239	
30.11 Constraint margins 241 30.12 The importance of opposing constraints and bias 242 30.13 Creating chains 245 30.14 Working with guidelines 246 30.15 Working with barriers 247 30.16 Decoupling constraints with constraint sets 250 30.17 Summary 252 31. Working with IntrinsicSize in Compose 253 31.1 Intrinsic measurements 253 31.2 Max. vs Min. Intrinsic Size measurements 253 31.3 About the example project 254 31.4 Creating the IntrinsicSizeDemo project 255 31.5 Creating the custom text field 255 31.6 Adding the Text and Box components 256 31.7 Adding the top-level Column 256 31.8 Testing the project. 257 31.9 Applying IntrinsicSize.Max measurements 257 31.10 Applying IntrinsicSize.Max measurements 257 32.1 Standard vs. lazy lists 259 32.2 Working with Column and Row lists 259 32.3 Creating lazy lists 259 32.4 Enabling scrolling with ScrollState 260 32.5 Programmatic scrolling 261		30.10 Constraint bias	240	
30.12 The importance of opposing constraints and bias 242 30.13 Creating chains 245 30.14 Working with guidelines 246 30.15 Working with barriers 247 30.16 Decoupling constraints with constraint sets 250 30.17 Summary 252 31. Working with IntrinsicSize in Compose 253 31.1 Intrinsic measurements 253 31.2 Max, vs Min. Intrinsic Size measurements 253 31.3 About the example project 254 31.4 Creating the IntrinsicSizeDemo project 255 31.5 Creating the custom text field 255 31.6 Adding the Text and Box components 256 31.7 Adding the top-level Column 256 31.8 Testing the project 257 31.9 Applying IntrinsicSize.Max measurements 257 31.10 Applying IntrinsicSize.Max measurements 258 32.1 Standard vs. lazy lists 259 32.2 Working with Column and Row lists 259 32.3 Creating lazy lists 260 32.4 Enabling scrolling with ScrollState 261 32.5 Programmatic scrolling 261 32.6 Sticky headers 263		30.11 Constraint margins	2/1	
30.12 Treating chains24230.13 Creating chains24530.14 Working with guidelines24630.15 Working with barriers24730.16 Decoupling constraints with constraint sets25030.17 Summary25231. Working with IntrinsicSize in Compose25331.1 Intrinsic measurements25331.2 Max. vs Min. Intrinsic Size measurements25331.3 About the example project25431.4 Creating the IntrinsicSizeDemo project25431.5 Creating the custom text field25531.6 Adding the Text and Box components25631.7 Adding the top-level Column25631.8 Testing the project25731.9 Applying IntrinsicSize.Max measurements25731.10 Applying IntrinsicSize.Min measurements25832. An Overview of Lists and Grids in Compose25932.1 Standard vs. lazy lists25932.2 Working with Column and Row lists25932.3 Creating lazy lists26032.4 Enabling scrolling with ScrollState26132.5 Programmatic scrolling26132.6 Sticky headers26332.7 Responding to scroll position26432.8 Creating a lazy grid26532.9 Summary267		30.12 The importance of opposing constraints and bias	241	
30.14 Working with guidelines24630.15 Working with barriers24730.16 Decoupling constraints with constraint sets25030.17 Summary25231. Working with IntrinsicSize in Compose25331.1 Intrinsic measurements25331.2 Max. vs Min. Intrinsic Size measurements25331.3 About the example project25431.4 Creating the IntrinsicSizeDemo project25531.5 Creating the custom text field25531.6 Adding the Text and Box components25631.7 Adding the top-level Column25631.8 Testing the project25731.9 Applying IntrinsicSize.Max measurements25831.11 Summary25832. An Overview of Lists and Grids in Compose25932.2 Working with Column and Row lists25932.3 Creating the scrolling with ScrollState26032.4 Enabling scrolling with ScrollState26132.5 Programmatic scrolling26132.6 Sticky headers26332.7 Responding to scroll position26432.8 Creating a lazy grid26532.9 Summary267		30.12 The importance of opposing constraints and bias	242	
30.14 Working with guidefines 240 30.15 Working with barriers 247 30.16 Decoupling constraints with constraint sets. 250 30.17 Summary 252 31. Working with IntrinsicSize in Compose. 253 31.1 Intrinsic measurements 253 31.2 Max. vs Min. Intrinsic Size measurements 253 31.3 About the example project 254 31.4 Creating the IntrinsicSizeDemo project. 255 31.5 Creating the custom text field. 255 31.6 Adding the Text and Box components 256 31.7 Adding the top-level Column 256 31.8 Testing the project. 257 31.9 Applying IntrinsicSize.Max measurements 257 31.10 Applying IntrinsicSize.Max measurements 258 32.1 Standard vs. lazy lists 259 32.2 Working with Column and Row lists 259 32.3 Creating lazy lists 259 32.4 Enabling scrolling with ScrollState 261 32.5 Programmatic scrolling 261 32.6 Sticky headers 263 32.7 Responding to scroll position 264 32.9 Summary 267		30.14 Working with guidelines	245	
30.15 Working with Obstraints with constraint sets.25030.17 Summary.25231. Working with IntrinsicSize in Compose.25331.1 Intrinsic measurements25331.2 Max. vs Min. Intrinsic Size measurements25331.3 About the example project25431.4 Creating the IntrinsicSizeDemo project.25531.5 Creating the custom text field.25531.6 Adding the Text and Box components25631.7 Adding the top-level Column25631.8 Testing the project.25731.9 Applying IntrinsicSize.Max measurements25731.10 Applying IntrinsicSize.Max measurements25832.1 Moverview of Lists and Grids in Compose25832.1 Standard vs. lazy lists25932.2 Working with Column and Row lists25932.3 Creating lazy lists25932.4 Enabling scrolling with ScrollState26132.5 Programmatic scrolling26132.6 Sticky headers26332.7 Responding to scroll position26432.8 Creating a lazy grid267		30.15 Working with barriers	240	
30.10 Decoupting constraints with constraint sets 250 30.17 Summary 252 31. Working with IntrinsicSize in Compose 253 31.1 Intrinsic measurements 253 31.2 Max, vs Min. Intrinsic Size measurements 253 31.3 About the example project 254 31.4 Creating the IntrinsicSizeDemo project. 255 31.5 Creating the custom text field 255 31.6 Adding the Text and Box components 256 31.7 Adding the top-level Column 256 31.8 Testing the project. 257 31.9 Applying IntrinsicSize.Max measurements 257 31.10 Applying IntrinsicSize.Max measurements 258 32.1 Norverview of Lists and Grids in Compose 259 32.2 Working with Column and Row lists 259 32.3 Creating lazy lists 260 32.4 Enabling scrolling, with ScrollState 261 32.5 Programmatic scrolling 261 32.6 Sticky headers 263 32.7 Responding to scroll position 264 32.9 Summary 265		30.16 Decoupling constraints with constraint sets	250	
30.17 Summary25231. Working with IntrinsicSize in Compose.25331.1 Intrinsic measurements.25331.2 Max. vs Min. Intrinsic Size measurements25331.3 About the example project.25431.4 Creating the IntrinsicSizeDemo project.25531.5 Creating the custom text field.25531.6 Adding the Text and Box components.25631.7 Adding the top-level Column25631.8 Testing the project.25731.9 Applying IntrinsicSize.Max measurements25731.10 Applying IntrinsicSize.Min measurements25832. An Overview of Lists and Grids in Compose25932.1 Standard vs. lazy lists25932.2 Working with Column and Row lists25932.3 Creating lazy lists26032.4 Enabling scrolling with ScrollState26132.5 Programmatic scrolling.26132.6 Sticky headers26332.7 Responding to scroll position26432.9 Summary267		30.17 Summary	250	
31.1 Intrinsic measurements25331.2 Max. vs Min. Intrinsic Size measurements25331.3 About the example project25431.4 Creating the IntrinsicSizeDemo project25531.5 Creating the custom text field25531.6 Adding the Text and Box components25631.7 Adding the top-level Column25631.8 Testing the project25731.9 Applying IntrinsicSize.Max measurements25731.10 Applying IntrinsicSize.Max measurements25831.11 Summary25832. An Overview of Lists and Grids in Compose25932.1 Standard vs. lazy lists25932.2 Creating lazy lists26032.4 Enabling scrolling with ScrollState26132.5 Programmatic scrolling26132.6 Sticky headers26332.7 Responding to scroll position26432.8 Creating a lazy grid26532.9 Summary267	31	Working with Intrinsic Size in Compose	232	253
31.1 Intrinsic measurements 253 31.2 Max. vs Min. Intrinsic Size measurements 253 31.3 About the example project 254 31.4 Creating the IntrinsicSizeDemo project. 255 31.5 Creating the custom text field. 256 31.7 Adding the Text and Box components 256 31.7 Adding the top-level Column. 256 31.8 Testing the project. 257 31.9 Applying IntrinsicSize.Max measurements 257 31.10 Applying IntrinsicSize.Max measurements 258 32. An Overview of Lists and Grids in Compose 259 32.1 Standard vs. lazy lists 259 32.2 Working with Column and Row lists 259 32.3 Creating lazy lists 260 32.4 Enabling scrolling with ScrollState 261 32.5 Programmatic scrolling 261 32.6 Sticky headers 263 32.7 Responding to scroll position 264 32.8 Creating a lazy grid 265 32.9 Summary 267	51.			255
31.2 Max, vs Min. Intrinsic Size measurements 253 31.3 About the example project 254 31.4 Creating the IntrinsicSizeDemo project 255 31.5 Creating the custom text field 255 31.6 Adding the Text and Box components 256 31.7 Adding the top-level Column 256 31.8 Testing the project 257 31.9 Applying IntrinsicSize.Max measurements 257 31.10 Applying IntrinsicSize.Max measurements 258 31.11 Summary 258 32. An Overview of Lists and Grids in Compose 259 32.1 Standard vs. lazy lists 259 32.3 Creating lazy lists 259 32.4 Enabling scrolling with ScrollState 261 32.5 Programmatic scrolling. 261 32.6 Sticky headers 263 32.7 Responding to scroll position 264 32.8 Creating a lazy grid 265 32.9 Summary 265		31.1 Intrinsic measurements	253	
31.3 About the example project. 254 31.4 Creating the IntrinsicSizeDemo project. 255 31.5 Creating the custom text field. 255 31.6 Adding the Text and Box components. 256 31.7 Adding the top-level Column. 256 31.8 Testing the project. 257 31.9 Applying IntrinsicSize.Max measurements 257 31.10 Applying IntrinsicSize.Min measurements 258 31.11 Summary 258 32.1 Standard vs. lazy lists 259 32.2 Working with Column and Row lists 259 32.3 Creating lazy lists 260 32.4 Enabling scrolling with ScrollState 261 32.5 Programmatic scrolling 261 32.6 Sticky headers 263 32.7 Responding to scroll position 264 32.8 Creating a lazy grid 265 32.9 Summary 265		31.2 Max. vs Min. Intrinsic Size measurements	253	
31.4 Creating the IntrinsicSizeDemo project. 255 31.5 Creating the custom text field. 255 31.6 Adding the Text and Box components 256 31.7 Adding the top-level Column. 256 31.8 Testing the project. 257 31.9 Applying IntrinsicSize.Max measurements 257 31.10 Applying IntrinsicSize.Min measurements 258 32. An Overview of Lists and Grids in Compose 259 32.1 Standard vs. lazy lists 259 32.2 Working with Column and Row lists 259 32.3 Creating lazy lists 260 32.4 Enabling scrolling with ScrollState 261 32.5 Programmatic scrolling 261 32.6 Sticky headers 263 32.7 Responding to scroll position 264 32.8 Creating a lazy grid 265 32.9 Summary 265		31.3 About the example project	254	
31.5 Creating the custom text field.25531.6 Adding the Text and Box components25631.7 Adding the top-level Column25631.8 Testing the project.25731.9 Applying IntrinsicSize.Max measurements25731.10 Applying IntrinsicSize.Min measurements25831.11 Summary25832. An Overview of Lists and Grids in Compose25932.1 Standard vs. lazy lists25932.2 Working with Column and Row lists25932.3 Creating lazy lists26032.4 Enabling scrolling with ScrollState26132.5 Programmatic scrolling26132.6 Sticky headers26332.7 Responding to scroll position26432.8 Creating a lazy grid26532.9 Summary267		31.4 Creating the IntrinsicSizeDemo project	255	
31.6 Adding the Text and Box components25631.7 Adding the top-level Column25631.8 Testing the project25731.9 Applying IntrinsicSize.Max measurements25731.10 Applying IntrinsicSize.Min measurements25831.11 Summary25832. An Overview of Lists and Grids in Compose25932.1 Standard vs. lazy lists25932.2 Working with Column and Row lists25932.3 Creating lazy lists26032.4 Enabling scrolling with ScrollState26132.5 Programmatic scrolling26132.6 Sticky headers26332.7 Responding to scroll position26432.8 Creating a lazy grid26532.9 Summary267		31.5 Creating the custom text field.	255	
31.7 Adding the top-level Column		31.6 Adding the Text and Box components	256	
31.8 Testing the project.25731.9 Applying IntrinsicSize.Max measurements25731.10 Applying IntrinsicSize.Min measurements25831.11 Summary25832. An Overview of Lists and Grids in Compose25932.1 Standard vs. lazy lists25932.2 Working with Column and Row lists25932.3 Creating lazy lists26032.4 Enabling scrolling with ScrollState26132.5 Programmatic scrolling26132.6 Sticky headers26332.7 Responding to scroll position26432.8 Creating a lazy grid26532.9 Summary267		31.7 Adding the top-level Column	256	
31.9 Applying IntrinsicSize.Max measurements25731.10 Applying IntrinsicSize.Min measurements25831.11 Summary25832. An Overview of Lists and Grids in Compose25932.1 Standard vs. lazy lists25932.2 Working with Column and Row lists25932.3 Creating lazy lists26032.4 Enabling scrolling with ScrollState26132.5 Programmatic scrolling26132.6 Sticky headers26332.7 Responding to scroll position26432.8 Creating a lazy grid26532.9 Summary267		31.8 Testing the project	257	
31.10 Applying IntrinsicSize.Min measurements25831.11 Summary25832. An Overview of Lists and Grids in Compose25932.1 Standard vs. lazy lists25932.2 Working with Column and Row lists25932.3 Creating lazy lists26032.4 Enabling scrolling with ScrollState26132.5 Programmatic scrolling26132.6 Sticky headers26332.7 Responding to scroll position26432.8 Creating a lazy grid26532.9 Summary267		31.9 Applying IntrinsicSize.Max measurements	257	
31.11 Summary		31.10 Applying IntrinsicSize.Min measurements	258	
32. An Overview of Lists and Grids in Compose25932.1 Standard vs. lazy lists25932.2 Working with Column and Row lists25932.3 Creating lazy lists26032.4 Enabling scrolling with ScrollState26132.5 Programmatic scrolling26132.6 Sticky headers26332.7 Responding to scroll position26432.8 Creating a lazy grid26532.9 Summary267		31.11 Summary	258	
32.1 Standard vs. lazy lists25932.2 Working with Column and Row lists25932.3 Creating lazy lists26032.4 Enabling scrolling with ScrollState26132.5 Programmatic scrolling26132.6 Sticky headers26332.7 Responding to scroll position26432.8 Creating a lazy grid26532.9 Summary267	32.	An Overview of Lists and Grids in Compose	•••••	259
32.2 Working with Column and Row lists25932.3 Creating lazy lists26032.4 Enabling scrolling with ScrollState26132.5 Programmatic scrolling26132.6 Sticky headers26332.7 Responding to scroll position26432.8 Creating a lazy grid26532.9 Summary267		32.1 Standard vs. lazy lists	259	
32.3 Creating lazy lists26032.4 Enabling scrolling with ScrollState26132.5 Programmatic scrolling26132.6 Sticky headers26332.7 Responding to scroll position26432.8 Creating a lazy grid26532.9 Summary267		32.2 Working with Column and Row lists	259	
32.4 Enabling scrolling with ScrollState26132.5 Programmatic scrolling26132.6 Sticky headers26332.7 Responding to scroll position26432.8 Creating a lazy grid26532.9 Summary267		32.3 Creating lazy lists	260	
32.5 Programmatic scrolling		32.4 Enabling scrolling with ScrollState	261	
32.6 Sticky headers 263 32.7 Responding to scroll position 264 32.8 Creating a lazy grid 265 32.9 Summary 267		32.5 Programmatic scrolling	261	
32.7 Responding to scroll position 264 32.8 Creating a lazy grid 265 32.9 Summary 267		32.6 Sticky headers	263	
32.8 Creating a lazy grid		32.7 Responding to scroll position	264	
32.9 Summary		32.8 Creating a lazy grid	265	
		32.9 Summary	267	

33. A Compose Row and Column List Tutorial	
33.1 Creating the ListDemo project	
33.2 Creating a Column-based list	
33.3 Enabling list scrolling	
33.4 Manual scrolling	
33.5 A Row list example	
33.6 Summary	274
24 A Compose Leave List Tyterial	
54. A Compose Lazy List futorial	
34.1 Creating the LazyListDemo project	
34.2 Adding list data to the project	
34.3 Reading the XML data	
34.4 Handling image loading	
34.5 Designing the list item composable	
34.6 Building the lazy list	
34.7 Testing the project	
34.8 Making list items clickable	
34.9 Summary	
35. Lazy List Sticky Headers and Scroll Detection	
35.1 Grouping the list item data	
35.2 Displaying the headers and items	285
35.3 Adding sticky headers	286
35.4 Reacting to scroll position	287
35.5 Adding the scroll button	289
35.6 Testing the finished app	291
35.7 Summary	
26 Compose Visibility Animation	202
36. Compose visibility Animation	
36.1 Creating the AnimateVisibility project	
36.2 Animating visibility	
36.3 Defining enter and exit animations	
36.4 Animation specs and animation easing	
36.5 Repeating an animation	
36.6 Different animations for different children	
36.7 Auto-starting an animation	
36.8 Implementing crossfading	
36.9 Summary	
37. Compose State-Driven Animation	
37.1 Understanding state-driven animation	
37.2 Introducing animate as state functions	
37.3 Creating the AnimateState project	
37.4 Animating rotation with animateFloatAsState	
37.5 Animating color changes with animateColorAsState	
37.6 Animating motion with animateDpAsState	
37.7 Adding spring effects	
37.8 Working with keyframes	
37.9 Combining multiple animations	
37.10 Using the Animation Inspector	
37.11 Summary	

	Canvas Graphics Drawing in Compose	
	38.1 Introducing the Canvas component	
	38.2 Creating the CanvasDemo project	
	38.3 Drawing a line and getting the canvas size	
	38.4 Drawing dashed lines	
	38.5 Drawing a rectangle	
	38.6 Applying rotation	325
	38.7 Drawing circles and ovals	326
	38.8 Drawing gradients	327
	38.9 Drawing arcs	330
	38.10 Drawing paths	331
	38.11 Drawing points	
	38.12 Drawing an image	333
	38.13 Summary	335
39.	Working with ViewModels in Compose	337
	20.1 What is Android Istnack?	227
	20.2 The "old" architecture	
	39.2 The old architecture	
	39.3 Modern Android architecture	
	39.4 The viewModel component.	
	39.5 ViewModel implementation using state	
	39.6 Connecting a viewiModel state to an activity	
	39.7 ViewModel implementation using LiveData.	
	39.8 Observing viewiviodel LiveData within an activity	
	39.9 Summary	
40.	A Compose ViewModel Tutorial	
	40.1 About the project	343
	40.1 About the project 40.2 Creating the ViewModelDemo project	343 344
	40.1 About the project	343 344 344
	 40.1 About the project 40.2 Creating the ViewModelDemo project 40.3 Adding the ViewModel 40.4 Accessing DemoViewModel from MainActivity 	343 344 344 345
	 40.1 About the project 40.2 Creating the ViewModelDemo project 40.3 Adding the ViewModel 40.4 Accessing DemoViewModel from MainActivity	343 344 344 345 346
	 40.1 About the project 40.2 Creating the ViewModelDemo project	343 344 344 345 346 347
	 40.1 About the project	
	 40.1 About the project	
	 40.1 About the project	
41.	 40.1 About the project	
41.	 40.1 About the project	
41.	 40.1 About the project	
41.	 40.1 About the project	
41.	 40.1 About the project	
41.	 40.1 About the project	
41.	 40.1 About the project	
41.	40.1 About the project	
41.	 40.1 About the project	
41.	40.1 About the project	
41.	40.1 About the project	
41.	40.1 About the project	
41.	40.1 About the project	

42.1 Repository 359 42.2.3 Data Access Object (DAO) 360 42.2.4 Entities 360 42.3 Understanding entities 361 42.4 Data Access Objects 363 42.5 The Room database 364 42.6 The Repository 365 42.7 In-Memory databases 366 42.8 Database Inspector 367 42.9 Summary 366 43.1 About the Room Dem project 369 43.1 About the RoomDemo project 370 43.5 Creating the RoomDemo project 370 43.6 Adding the Room database 373 43.7 Adding the repository 374 43.8 Adding the WieWodel 376 43.9 Designing the user interface 378 43.10 Completing the MainScreen function 380 43.11 Testing the RoomDemo app 383 43.12 Using the Database Inspector 383 43.13 Summary 384 44.1 An Overeview of Navigation in Compose 387		42.2 Key elements of Room database persistence	359
42.2 2 Room database 360 42.2.3 Data Access Object (DAO) 360 42.2.4 Entities 360 42.2.5 SQLite database 360 42.2.5 SQLite database 361 42.1 Data Access Objects 363 42.2.5 The Room database 364 42.6 The Repository 365 42.7 In -Memory databases 366 42.8 Database Inspector 367 42.9 Summary 367 43. A Compose Room Database and Repository Tutorial 369 43.1 About the RoomDemo project 370 43.4 Building the build configuration 370 43.5 Creating the Data Access Object 372 43.6 Adding the repository. 374 43.7 Adding the view Model 376 43.9 Designing the user interface 376 43.10 Completing the MainScreen function 380 43.11 Testing the RoomDemo app 382 43.13 13 Summary 383 43.12 Using the Database Inspector 383 43.12 Using the Database Inspector 381 43.13 Completing the MainScreen function 387 44.1 An Overview of Navigation in Compose		42.2.1 Repository	359
42.23 Data Access Object (DAO) 360 42.24 Entities 360 42.2.4 Entities 360 42.3 Understanding entities 361 42.4 Data Access Objects 363 42.5 The Room database 364 42.6 The Repository 365 42.7 In-Memory databases 366 42.7 In-Memory databases 366 42.7 In-Memory databases 366 42.8 Database Inspector 367 42.9 Summary 367 43.1 About the RoomDemo project 369 43.1 About the RoomDemo project 370 43.3 Modifying the build configuration 370 43.4 Building the entity 370 43.5 Creating the Room database 373 43.7 Adding the ViewModel 376 43.9 Designing the user interface 378 43.10 Completing the ManScreen function 380 43.11 Testing the RoomDemo app 383 43.13 Summary 384 44.4 An Overview of Navigation ont Compose 385 44.1 Understanding navigation markingtion graph 387 44.3 Declaring a navigation onetoller 387 <		42.2.2 Room database	360
42.2.4 Entities 360 42.2.5 SQLite database 360 42.2.5 SQLite database 361 42.4 Data Access Objects 363 42.5 The Room database 364 42.6 The Repository 365 42.7 In-Memory databases 366 42.7 In-Memory databases 367 42.9 Summary 367 42.9 Summary 367 43.1 About the RoomDemo project 369 43.1 About the RoomDemo project 370 43.4 Building the entity 370 43.5 Creating the Room Database 372 43.6 Adding the Pository 372 43.6 Adding the Room database 373 43.7 Adding the repository 374 43.8 Adding the ViewModel 376 43.9 Designing the user interface 376 43.10 Coupleting the MainScreen function 380 43.11 Testing the RoomDemo app 383 43.12 Using the Database Inspector 383 43.12 Using the Database Inspector 383 43.12 Using the Database Inspector 383 44.1 Understanding navigation in Compose 387 44.		42.2.3 Data Access Object (DAO)	360
42.2.5 SQLite database 360 42.3 Understanding entities 361 42.4 Data Access Objects 363 42.5 The Room database 364 42.6 The Repository 365 42.7 In -Memory databases 366 42.8 Database Inspector. 367 42.9 Summary 367 43.1 About the RoomDemo project 369 43.2 Creating the RoomDemo project. 370 43.3 Modifying the build configuration 370 43.4 Building the entity. 370 43.5 Creating the Data Access Object. 372 43.6 Adding the RoomDemo project. 370 43.7 Adding the repository. 374 43.8 Adding the ViewModel 373 43.7 Adding the repository. 374 43.8 Adding the NornDemo app. 383 43.10 Completing the MainScreen function. 380 43.11 Testing the RoomDemo app. 383 43.12 Summary. 384 44.10 Understanding navigation in Compose. 385 44.11 Understanding navigation. 387 44.3 Declaring a navigation notroller. 387 44.4 Ado passing arguments to a destinat		42.2.4 Entities	360
42.3 Understanding entities 361 42.4 Data Access Objects 363 42.5 The Room database 364 42.6 The Room database 365 42.7 In-Memory databases 366 42.8 Database Inspector 367 42.9 Summary 367 42.9 Summary 367 43.4 Compose Room Database and Repository Tutorial 369 43.1 About the RoomDemo project 370 43.3 Modifying the build configuration 370 43.4 Building the entity 370 43.5 Creating the Room database 372 43.6 Adding the Room database 373 43.7 Adding the repository 374 43.8 Adding the ViewModel 376 43.9 Designing the user interface 378 43.10 Completing the MainScreen function 380 43.11 Esting the RoomDemo app 383 43.12 Using the Database Inspector 383 43.13 Summary 384 44. An Overview of Navigation in Compose 387 44.1 Understanding navigation nottroller 387 44.2 Declaring a navigation host 388 44.4 Adding destinations. <		42.2.5 SQLite database	360
42.4 Data Access Öbjects. 363 42.5 The Room database 364 42.6 The Repository 365 42.7 In-Memory databases. 366 42.8 Database Inspector. 367 42.8 Database Inspector. 367 42.9 Summary. 367 43.1 About the RoomDemo project. 369 43.1 About the RoomDemo project. 370 43.3 Modifying the build configuration 370 43.4 Building the entity. 370 43.5 Creating the Data Access Object. 372 43.6 Adding the Provisiory. 373 43.7 Adding the repository. 374 43.8 Adding the ViewModel 376 43.9 Designing the user interface 378 43.10 Completing the MainScreen function. 380 43.11 Summary. 384 44. An Overview of Navigation in Compose 387 44.1 Understanding navigation. 385 44.1 Understanding navigation controller. 387 44.5 Navigating to destination. 390 45.1 Creating the Navigation bars 381 44.4 Adding destinations to the navigation project. 395 45.1 Overi		42.3 Understanding entities	361
42.5 The Room database36442.6 The Repository36542.7 In-Memory databases36642.8 Database Inspector36742.9 Summary36743. A Compose Room Database and Repository Tutorial36943.1 About the RoomDemo project36943.2 Creating the RoomDemo project37043.3 Modifying the build configuration37043.4 Building the entity37043.5 Creating the Data Access Object37243.6 Adding the Room database37343.7 Adding the repository37443.8 Adding the ViewModel37643.9 Designing the user interface37843.10 Completing the MainScreen function38043.11 Testing the RoomDemo app38343.12 Using the Database Inspector38343.13 Summary38444. An Overview of Navigation in Compose38544.1 Understanding navigation.38544.2 Declaring a navigation controller38744.8 Summary39144.8 Summary39345. A Compose Navigation Tutorial39545.1 Creating the Navigation bost38744.3 Adding destinations to the navigation project39545.1 Creating the Navigation Demo project39545.1 Creating the Navigation Demo project39545.3 Declaring the navigation Demo project39545.4 Adding the travigation Demo project39545.4 Adding the nove screen39645.5 Adding the envigation Controller and host39945.6 Adding		42.4 Data Access Objects	363
42.6 The Repository.36542.7 In-Memory databases36642.8 Database Inspector.36742.9 Summary.36743. A Compose Room Database and Repository Tutorial.36943.1 About the RoomDemo project.36943.2 Creating the RoomDemo project.37043.3 Modifying the build configuration37043.4 Building the entity.37043.5 Creating the Data Access Object.37243.6 Adding the Room database37343.7 Adding the repository.37443.8 Adding the ViewModel37643.9 Designing the use interface37843.10 Completing the MainScreen function38043.11 Testing the RoomDemo app38343.12 Using the Database Inspector.38444. An Overview of Navigation in Compose38744.1 Understanding navigation not ontroller.38744.2 Declaring a navigation host38744.3 Declaring a navigation host38744.4 Adding destinations.38844.4 Adding to destination.39044.7 Working with bottom navigation bars39144.8 Summary.39344.8 Summary.39345. A Compose Navigation Tutorial39745. A Compose Navigation		42.5 The Room database	364
42.7 In-Memory databases 366 42.8 Database Inspector 367 42.9 Summary 367 43. A Compose Room Database and Repository Tutorial 369 43.1 About the RoomDemo project 369 43.2 Creating the RoomDemo project 370 43.3 Modifying the build configuration 370 43.4 Building the entity 370 43.5 Creating the Data Access Object 372 43.6 Adding the repository 372 43.6 Adding the repository 374 43.8 Adding the viewModel 376 43.9 Designing the user interface 378 43.10 Completing the MainScreen function 380 43.11 Testing the RoomDemo app 383 43.13 Summary 384 44. An Overview of Navigation in Compose 385 44.1 Understanding navigation notroller 387 44.3 Declaring a navigation host 387 44.4 Ading destinations. 386 45.1 Creating the Navigation Event 387 44.4 Ading destinations to the navigation graph 387 44.3 Declaring a navigation host 387 45.4 Creating the NavigationDemo project 395		42.6 The Repository	365
42.8 Database Inspector 367 42.9 Summary 367 43. A Compose Room Database and Repository Tutorial 369 43.1 About the RoomDemo project 369 43.2 Creating the RoomDemo project 370 43.3 Modifying the build configuration 370 43.4 Building the entity 370 43.5 Creating the Data Access Object 372 43.6 Adding the Room database 373 43.7 Adding the repository 374 43.8 Adding the ViewModel 376 43.9 Designing the user interface 378 43.10 Completing the MainScreen function 380 43.11 Testing the RoomDemo app 383 43.13 Summary 384 44. An Overview of Navigation in Compose 385 44.1 Understanding navigation controller 387 44.3 Declaring a navigation host 387 44.4 Adding destinations 388 44.6 Passing arguments to a destination 390 45. A Compose Navigation Tutorial 397 45. A Compose Navigation Tutorial 395 45.1 Creating the NavigationDemo project 395 45.2 About the NavigationDemo project		42.7 In-Memory databases	366
42.9 Summary 367 43. A Compose Room Database and Repository Tutorial 369 43.1 About the RoomDemo project 369 43.2 Creating the Room Demo project 370 43.3 Modifying the build configuration 370 43.4 Building the entity. 370 43.5 Creating the Data Access Object 372 43.6 Adding the repository 374 43.8 Adding the repository 374 43.8 Adding the viewModel 376 43.9 Designing the user interface 378 43.10 Completing the MainScreen function 380 43.11 Testing the RoomDemo app 383 43.12 Using the Database Inspector 383 43.13 Summary 384 44. An Overview of Navigation in Compose 385 44.1 Duderstanding navigation controller 387 44.3 Declaring a avigation host 387 44.4 Ading destinations to the navigation graph 387 44.5 Navigating to destinations 388 44.6 Passing arguments to a destination 390 47.7 Working with bottom navigation bars 391 44.8 Summary 393 45. A Compose Navigation Demo project<		42.8 Database Inspector	367
43. A Compose Room Database and Repository Tutorial 369 43.1 About the RoomDemo project 369 43.2 Creating the RoomDemo project 370 43.3 Modifying the build configuration 370 43.4 Building the entity 370 43.5 Creating the Data Access Object 372 43.6 Adding the repository 373 43.7 Adding the repository 374 43.8 Adding the ViewModel 376 43.9 Designing the user interface 378 43.10 Completing the MainScreen function 380 43.11 Testing the RoomDemo app 383 43.12 Using the Database Inspector 383 43.13 Summary 384 44. An Overview of Navigation in Compose 385 44.1 Understanding navigation 385 44.2 Declaring a navigation host 387 44.3 Declaring a ravigation tothe navigation graph 387 44.4 Ading destinations 390 44.7 Working with bottom navigation bars 391 44.8 Ausumary 393 45. A Compose Navigation Tutorial 395 45. A Corpose Navigation Tutorial 395 45. A Creating the NavigationDemo proje		42.9 Summary	367
43.1 About the RoomDemo project.36943.2 Creating the RoomDemo project.37043.3 Modifying the build configuration37043.4 Building the entity37043.5 Creating the Data Access Object.37243.6 Adding the Room database.37343.7 Adding the repository.37443.8 Adding the ViewModel37643.9 Designing the user interface37843.10 Completing the MainScreen function38043.11 Testing the RoomDemo app.38343.12 Using the Database Inspector38343.13 Summary.38444. An Overview of Navigation in Compose38744.4 Adding destinations to the navigation graph38744.5 Navigating to destinations.38744.6 Passing arguments to a destination.39044.7 Working with bottom navigation bars39144.8 Summary.39345. A Compose Navigation Tutorial39545.1 Creating the NavigationDemo project.39545.2 About the NavigationDemo project.39545.3 Declaring the navigation notes39545.4 Adding the inavigation notes39545.5 Adding the navigation notes39545.6 Adding the navigation notes39645.7 Creating the NavigationDemo project.39545.8 Implementing the screen39645.7 Creating the navigation notes39545.8 Implementing the screen39745.8 Ming the view contex screen39645.7 Creating the navigation controller and host39945	43.	A Compose Room Database and Repository Tutorial	369
43.2 Creating the RoomDemo project.37043.3 Modifying the build configuration37043.4 Building the entity37043.4 Building the entity37043.5 Creating the Data Access Object.37243.6 Adding the Room database.37343.7 Adding the repository.37443.8 Adding the repository.37443.8 Adding the viewModel37643.9 Designing the user interface.37843.10 Completing the MainScreen function.38043.11 Testing the RoomDemo app.38343.12 Using the Database Inspector38343.13 Summary.38444. An Overview of Navigation in Compose38544.1 Understanding navigation.38544.2 Declaring a navigation notroller.38744.3 Declaring a navigation notroller.38744.4 Adding destinations to the navigation graph38744.6 Passing arguments to a destination.39044.7 Working with bottom navigation bars.39144.8 Summary.39345. A Compose Navigation Tutorial39545.1 Creating the NavigationDemo project.39545.3 Declaring the navigation controller and host.39945.4 Adding the wavigation controller and host.39945.5 Adding the profile screen39645.6 Adding the profile screen39645.7 Creating the navigation controller and host.39945.8 Implementing the screen navigation.39945.9 Passing the user name argument.40045.11 Summary.403 <td></td> <td>43.1 About the RoomDemo project</td> <td> 369</td>		43.1 About the RoomDemo project	369
43.3 Modifying the build configuration37043.4 Building the entity		43.2 Creating the RoomDemo project	370
43.4 Building the entity.37043.5 Creating the Data Access Object.37243.6 Adding the Room database37343.7 Adding the repository.37443.8 Adding the ViewModel37643.9 Designing the user interface.37843.10 Completing the MainScreen function.38043.11 Testing the RoomDemo app.38343.12 Using the Database Inspector.38343.13 Summary.38444. An Overview of Navigation in Compose38544.1 Understanding navigation.38544.2 Declaring a navigation notroller.38744.4 Adding destinations to the navigation graph.38744.5 Navigating to destinations.38844.6 Passing arguments to a destination.39044.7 Working with bottom navigation bars.39145. A Compose Navigation Tutorial39545.1 Creating the NavigationDemo project.39545.2 About the NavigationDemo project.39545.3 Declaring the navigation notes39545.4 Adding the invagation routes39545.5 Adding the navigation notes39545.6 Adding the navigation notes39545.7 Creating the navigation notes39545.6 Adding the rofile screen39645.7 Creating the navigation controller and host39945.8 Implementing the screen navigation39945.9 Passing the user name argument40045.10 Testing the project40145.11 Summary.403		43.3 Modifying the build configuration	
43.5 Creating the Data Access Object37243.6 Adding the Room database37343.7 Adding the Room database37343.8 Adding the ViewModel37643.9 Designing the user interface37843.10 Completing the MainScreen function38043.11 Testing the RoomDemo app38343.12 Using the Database Inspector38343.13 Summary38444. An Overview of Navigation in Compose38544.1 Understanding navigation38544.2 Declaring a navigation controller38744.3 Declaring a navigation not to the navigation graph38744.4 Adding destinations38744.5 Navigating to destinations38744.6 Passing arguments to a destination39044.7 Working with bottom navigation bars39145. A Compose Navigation Tutorial39545.1 Creating the NavigationDemo project39545.3 Declaring the navigation notes39545.4 Adding the registion Tutorial39545.5 Adding the navigation notes39545.6 Adding the profile screen39645.7 Creating the navigation notes39545.6 Adding the profile screen39645.7 Creating the navigation controller and host39945.8 Implementing the screen navigation39945.9 Passing the user name argument40045.10 Testing the project40145.11 Summary403		43 4 Building the entity	370
43.6 Adding the Room database.37343.7 Adding the Room database.37443.8 Adding the repository.37443.8 Adding the ViewModel37643.9 Designing the user interface37843.10 Completing the MainScreen function.38043.11 Completing the RoomDemo app.38343.12 Using the Database Inspector.38343.13 Summary.38444. An Overview of Navigation in Compose38544.1 Understanding navigation38544.2 Declaring a navigation host38744.3 Declaring a navigation host38744.4 Adding destinations to the navigation graph38744.5 Navigating to destinations.38844.6 Passing arguments to a destination.39044.7 Working with bottom navigation bars39144.8 Summary.39345. A Compose Navigation Duemo project39545.1 Creating the NavigationDemo project39545.3 Declaring the navigation routes39545.4 Adding the hone screen39645.5 Adding the navigation controller and host39945.6 Adding the hore screen39645.7 Creating the navigation controller and host39945.8 Implementing the screen navigation39945.9 Passing the user name argument40045.10 Testing the project40145.11 Summary403		43.5 Creating the Data Access Object	372
43.7 Adding the repository.37443.8 Adding the ViewModel37643.9 Designing the user interface37843.10 Completing the MainScreen function38043.11 Testing the RoomDemo app38343.12 Using the Database Inspector38343.13 Summary38444. An Overview of Navigation in Compose38544.1 Understanding navigation38544.2 Declaring a navigation controller38744.3 Declaring a navigation host38744.4 Adding destinations to the navigation graph38744.5 Navigating to destinations38844.6 Passing arguments to a destination39044.7 Working with bottom navigation bars39144.8 Summary39345. A Compose NavigationDemo project39545.1 Creating the NavigationDemo project39545.2 About the NavigationDemo project39545.3 Declaring the navigation routes39545.4 Adding the home screen39645.5 Adding the velocem screen39745.6 Adding the profile screen39745.6 Adding the profile screen39845.7 Creating the navigation controller and host39945.9 Passing the user name argument40045.10 Testing the project40145.11 Summary403		43.6 Adding the Room database	373
43.8 Adding the ViewModel37643.9 Designing the user interface37843.10 Completing the MainScreen function38043.11 Testing the RoomDemo app38343.12 Using the Database Inspector38343.13 Summary38444. An Overview of Navigation in Compose38544.1 Understanding navigation38544.2 Declaring a navigation controller38744.3 Declaring a navigation not the navigation graph38744.4 Adding destinations to the navigation graph38744.5 Navigating to destinations38644.6 Passing arguments to a destination39044.7 Working with bottom navigation bars39144.8 Summary39345. A Compose Navigation Tutorial39545.1 Creating the NavigationDemo project39545.2 About the NavigationDemo project39545.3 Declaring the navigation controller and host39945.6 Adding the profile screen39645.7 Creating the navigation controller and host39945.8 Implementing the screen navigation39945.9 Passing the user name argument40045.11 Summary403		43.7 Adding the repository	374
43.9 Designing the user interface37843.10 Completing the MainScreen function38043.11 Testing the RoomDemo app38343.12 Using the Database Inspector38343.13 Summary38444. An Overview of Navigation in Compose38544.1 Understanding navigation38544.2 Declaring a navigation controller38744.3 Declaring a navigation host38744.4 Adding destinations to the navigation graph38744.5 Navigating to destinations38844.6 Passing arguments to a destination39044.7 Working with bottom navigation bars39144.8 Summary39345. A Compose Navigation Demo project39545.1 Creating the NavigationDemo project39545.2 About the NavigationDemo project39545.3 Declaring the NavigationDemo project39545.4 Adding the home screen39645.5 Adding the kelcome screen39745.6 Adding the profile screen39845.7 Creating the navigation controller and host39945.8 Implementing the screen navigation39945.9 Passing the user name argument40045.10 Testing the project40145.11 Summary403		43.8 Adding the ViewModel	376
43.10 Completing the MainScreen function38043.11 Testing the RoomDemo app38343.12 Using the Database Inspector38343.13 Summary38444. An Overview of Navigation in Compose38544.1 Understanding navigation38544.2 Declaring a navigation controller38744.3 Declaring a navigation host38744.4 Adding destinations to the navigation graph38744.5 Navigating to destinations38844.6 Passing arguments to a destination39044.7 Working with bottom navigation bars39144.8 Summary39345. A Compose Navigation Tutorial39545.1 Creating the NavigationDemo project39545.2 About the NavigationDemo project39545.3 Declaring the navigation routes39645.5 Adding the home screen39745.6 Adding the home screen39745.6 Adding the user name argument.40045.10 Testing the navigation controller and host39945.10 Testing the project40145.11 Summary403		13.9 Designing the user interface	378
43.11 Testing the NamoEur function36043.11 Testing the RoomDemo app.38343.12 Using the Database Inspector38343.13 Summary38444. An Overview of Navigation in Compose38544.1 Understanding navigation38544.2 Declaring a navigation controller38744.3 Declaring a navigation host38744.4 Adding destinations to the navigation graph38744.5 Navigating to destinations38844.6 Passing arguments to a destination39044.7 Working with bottom navigation bars39144.8 Summary39345. A Compose Navigation Tutorial39545.1 Creating the NavigationDemo project39545.2 About the NavigationDemo project39545.5 Adding the home screen39645.5 Adding the welcome screen39745.6 Adding the profile screen39845.7 Creating the navigation controller and host39945.9 Passing the user name argument.40045.10 Testing the project40145.11 Summary403		43.10 Completing the MainScreen function	380
43.12 Using the Database Inspector.38343.13 Summary.38444. An Overview of Navigation in Compose38544.1 Understanding navigation.38544.2 Declaring a navigation controller.38744.3 Declaring a navigation host38744.4 Adding destinations to the navigation graph.38744.5 Navigating to destinations.38844.6 Passing arguments to a destination.39044.7 Working with bottom navigation bars39144.8 Summary.39345. A Compose Navigation Tutorial39545.1 Creating the NavigationDemo project39545.2 About the NavigationDemo project39545.3 Declaring the navigation routes39545.4 Adding the home screen39645.5 Adding the velcome screen39745.6 Adding the profile screen39845.7 Creating the navigation controller and host39945.8 Implementing the screen navigation39945.9 Passing the user name argument.40045.11 Summary.403		43.11 Testing the RoomDemo app	383
43.13 Comparison38343.13 Summary38444. An Overview of Navigation in Compose38544.1 Understanding navigation38544.2 Declaring a navigation controller38744.3 Declaring a navigation host38744.4 Adding destinations to the navigation graph38744.5 Navigating to destinations38844.6 Passing arguments to a destination39044.7 Working with bottom navigation bars39144.8 Summary39345.1 Creating the NavigationDemo project39545.1 Creating the NavigationDemo project39545.3 Declaring the navigation routes39545.4 Adding the home screen39645.5 Adding the welcome screen39745.6 Adding the profile screen39745.7 Creating the navigation controller and host39945.8 Implementing the screen navigation39945.9 Passing the user name argument40045.10 Testing the project40145.11 Summary403		43.12 Using the Database Inspector	303
44. An Overview of Navigation in Compose38544.1 Understanding navigation.38544.2 Declaring a navigation controller.38744.3 Declaring a navigation host38744.4 Adding destinations to the navigation graph38744.5 Navigating to destinations.38844.6 Passing arguments to a destination.39044.7 Working with bottom navigation bars39144.8 Summary39345. A Compose Navigation Tutorial39545.1 Creating the NavigationDemo project.39545.3 Declaring the navigation routes39545.4 Adding the home screen39645.5 Adding the welcome screen39745.6 Adding the profile screen39745.7 Creating the navigation controller and host39945.8 Implementing the screen navigation39945.9 Passing the user name argument40045.10 Testing the project40145.11 Summary403		43.13 Summary	384
44.1 Understanding navigation38544.2 Declaring a navigation controller38744.3 Declaring a navigation host38744.4 Adding destinations to the navigation graph38744.5 Navigating to destinations38844.6 Passing arguments to a destination39044.7 Working with bottom navigation bars39144.8 Summary39345. A Compose Navigation Tutorial39545.1 Creating the NavigationDemo project39545.2 About the NavigationDemo project39545.3 Declaring the navigation routes39545.4 Adding the home screen39645.5 Adding the welcome screen39745.6 Adding the profile screen39845.7 Creating the navigation controller and host39945.8 Implementing the screen navigation39945.9 Passing the user name argument40045.10 Testing the project40145.11 Summary403	44.	An Overview of Navigation in Compose	
44.1 Onderstanding navigation36344.2 Declaring a navigation controller38744.3 Declaring a navigation host38744.4 Adding destinations to the navigation graph38744.5 Navigating to destinations.38844.6 Passing arguments to a destination39044.7 Working with bottom navigation bars39144.8 Summary39345. A Compose Navigation Tutorial39545.1 Creating the NavigationDemo project39545.2 About the NavigationDemo project39545.3 Declaring the navigation routes39545.4 Adding the home screen39645.5 Adding the welcome screen39745.6 Adding the profile screen39845.7 Creating the navigation controller and host39945.8 Implementing the screen navigation39945.9 Passing the user name argument40045.10 Testing the project40145.11 Summary403		14.1 Understanding navigation	295
44.2 Declaring a navigation controller38744.3 Declaring a navigation host38744.4 Adding destinations to the navigation graph38744.5 Navigating to destinations38844.6 Passing arguments to a destination39044.7 Working with bottom navigation bars39144.8 Summary39345. A Compose Navigation Tutorial39545.1 Creating the NavigationDemo project39545.2 About the NavigationDemo project39545.3 Declaring the navigation routes39545.5 Adding the home screen39645.5 Adding the welcome screen39745.6 Adding the profile screen39845.7 Creating the navigation controller and host39945.8 Implementing the screen navigation39945.9 Passing the user name argument40045.11 Summary403		44.1 Onderstanding navigation	303
44.3 Dectaring a navigation nost38744.4 Adding destinations to the navigation graph38744.5 Navigating to destinations38844.6 Passing arguments to a destination39044.7 Working with bottom navigation bars39144.8 Summary39345. A Compose Navigation Tutorial39545.1 Creating the NavigationDemo project39545.2 About the NavigationDemo project39545.3 Declaring the navigation routes39645.5 Adding the home screen39645.5 Adding the welcome screen39745.6 Adding the profile screen39845.7 Creating the navigation controller and host39945.8 Implementing the screen navigation39945.9 Passing the user name argument40045.10 Testing the project40145.11 Summary403		44.2 Declaring a navigation controller	387
44.4 Adding destinations to the navigation graph38744.5 Navigating to destinations38844.6 Passing arguments to a destination39044.7 Working with bottom navigation bars39144.8 Summary39345. A Compose Navigation Tutorial39545.1 Creating the NavigationDemo project39545.2 About the NavigationDemo project39545.3 Declaring the navigation routes39545.4 Adding the home screen39645.5 Adding the welcome screen39745.6 Adding the profile screen39845.7 Creating the navigation controller and host39945.8 Implementing the screen navigation39945.9 Passing the user name argument40045.10 Testing the project40145.11 Summary403		44.5 Declaring a navigation nost	387
44.5 Navigating to destinations		44.4 Adding destinations to the navigation graph	387
44.6 Passing arguments to a destination39044.7 Working with bottom navigation bars39144.8 Summary39345. A Compose Navigation Tutorial39545.1 Creating the NavigationDemo project39545.2 About the NavigationDemo project39545.3 Declaring the navigation routes39545.4 Adding the home screen39645.5 Adding the welcome screen39745.6 Adding the profile screen39845.7 Creating the navigation controller and host39945.8 Implementing the screen navigation39945.9 Passing the user name argument40045.10 Testing the project40145.11 Summary403		44.5 Navigating to destinations.	388
44.7 Working with bottom navigation bars39144.8 Summary39345. A Compose Navigation Tutorial39545.1 Creating the NavigationDemo project39545.2 About the NavigationDemo project39545.3 Declaring the navigation routes39545.4 Adding the home screen39645.5 Adding the welcome screen39745.6 Adding the profile screen39845.7 Creating the navigation controller and host39945.8 Implementing the screen navigation39945.9 Passing the user name argument40045.10 Testing the project40145.11 Summary403		44.6 Passing arguments to a destination.	390
44.8 Summary39345. A Compose Navigation Tutorial39545.1 Creating the NavigationDemo project39545.2 About the NavigationDemo project39545.3 Declaring the navigation routes39545.4 Adding the home screen39645.5 Adding the welcome screen39745.6 Adding the profile screen39845.7 Creating the navigation controller and host39945.8 Implementing the screen navigation39945.9 Passing the user name argument40045.10 Testing the project40145.11 Summary403		44.7 Working with bottom navigation bars	391
45. A Compose Navigation Tutorial39545.1 Creating the NavigationDemo project39545.2 About the NavigationDemo project39545.3 Declaring the navigation routes39545.4 Adding the home screen39645.5 Adding the welcome screen39745.6 Adding the profile screen39845.7 Creating the navigation controller and host39945.8 Implementing the screen navigation39945.9 Passing the user name argument40045.10 Testing the project40145.11 Summary403		44.8 Summary	393
45.1 Creating the NavigationDemo project39545.2 About the NavigationDemo project39545.3 Declaring the navigation routes39545.4 Adding the home screen39645.5 Adding the welcome screen39745.6 Adding the profile screen39845.7 Creating the navigation controller and host39945.8 Implementing the screen navigation39945.9 Passing the user name argument40045.10 Testing the project40145.11 Summary403	45.	A Compose Navigation Tutorial	
45.2 About the NavigationDemo project39545.3 Declaring the navigation routes39545.4 Adding the home screen39645.5 Adding the welcome screen39745.6 Adding the profile screen39845.7 Creating the navigation controller and host39945.8 Implementing the screen navigation39945.9 Passing the user name argument40045.10 Testing the project40145.11 Summary403		45.1 Creating the NavigationDemo project	395
45.3 Declaring the navigation routes39545.4 Adding the home screen39645.5 Adding the welcome screen39745.6 Adding the profile screen39845.7 Creating the navigation controller and host39945.8 Implementing the screen navigation39945.9 Passing the user name argument40045.10 Testing the project40145.11 Summary403		45.2 About the NavigationDemo project	395
45.4 Adding the home screen39645.5 Adding the welcome screen39745.6 Adding the profile screen39845.7 Creating the navigation controller and host39945.8 Implementing the screen navigation39945.9 Passing the user name argument40045.10 Testing the project40145.11 Summary403		45.3 Declaring the navigation routes	395
45.5 Adding the welcome screen39745.6 Adding the profile screen39845.7 Creating the navigation controller and host39945.8 Implementing the screen navigation39945.9 Passing the user name argument40045.10 Testing the project40145.11 Summary403		45.4 Adding the home screen	396
45.6 Adding the profile screen39845.7 Creating the navigation controller and host39945.8 Implementing the screen navigation39945.9 Passing the user name argument40045.10 Testing the project40145.11 Summary403		45.5 Adding the welcome screen	397
45.7 Creating the navigation controller and host39945.8 Implementing the screen navigation39945.9 Passing the user name argument40045.10 Testing the project40145.11 Summary403		45.6 Adding the profile screen	398
45.8 Implementing the screen navigation39945.9 Passing the user name argument40045.10 Testing the project40145.11 Summary403		45.7 Creating the navigation controller and host	399
45.9 Passing the user name argument		45.8 Implementing the screen navigation	399
45.10 Testing the project		45.9 Passing the user name argument	400
45.11 Summary		45.10 Testing the project	401
		45.11 Summary	403

46 .	A Compose Bottom Navigation Bar Tutorial	•••••	405
	46.1 Creating the BottomBarDemo project	405	
	46.2 Declaring the navigation routes	405	
	46.3 Designing bar items	406	
	46.4 Creating the bar item list	406	
	46.5 Adding the destination screens	407	
	46.6 Creating the navigation controller and host	409	
	46.7 Designing the navigation bar	410	
	46.8 Working with the Scaffold component	411	
	46.9 Testing the project	412	
	46.10 Summary	412	
47.	Detecting Gestures in Compose		413
	47.1 Compose gesture detection	413	
	47.2 Creating the Gesture Demo project	413	
	47.3 Detecting click gestures	413	
	47.4 Detecting tans using PointerInnutScope	415	
	47.5 Detecting drag gestures	416	
	47.6 Detecting drag gestures using PointerInputScope	418	
	47.7 Scrolling using the scrollable modifier	419	
	47.8 Scrolling using the scroll modifiers	420	
	47.9 Detecting ninch gestures	420	
	47.10 Detecting rotation gestures	423	
	47 11 Detecting translation gestures	424	
	47.12 Summary		
48	Detecting Swine Gestures in Compose		427
48.	Detecting Swipe Gestures in Compose	427	427
48.	Detecting Swipe Gestures in Compose	427	427
48.	Detecting Swipe Gestures in Compose	427 427 428	427
48.	Detecting Swipe Gestures in Compose	427 427 428 428	427
48.	Detecting Swipe Gestures in Compose 48.1 Swipe gestures and anchors 48.2 Detecting swipe gestures 48.3 Declaring the anchors map 48.4 Declaring thresholds 48.5 Maxing a component in response to a giving	427 427 428 428 428	427
48.	Detecting Swipe Gestures in Compose 48.1 Swipe gestures and anchors 48.2 Detecting swipe gestures 48.3 Declaring the anchors map 48.4 Declaring thresholds 48.5 Moving a component in response to a swipe 48.6 About the SwipeDemo project		427
48.	Detecting Swipe Gestures in Compose 48.1 Swipe gestures and anchors 48.2 Detecting swipe gestures 48.3 Declaring the anchors map 48.4 Declaring thresholds 48.5 Moving a component in response to a swipe 48.6 About the SwipeDemo project 48.7 Creating the SwipeDemo project		427
48.	Detecting Swipe Gestures in Compose 48.1 Swipe gestures and anchors 48.2 Detecting swipe gestures 48.3 Declaring the anchors map 48.4 Declaring thresholds 48.5 Moving a component in response to a swipe 48.6 About the SwipeDemo project 48.7 Creating the SwipeDemo project 48.8 Setting the SwipeDemo project		427
48.	Detecting Swipe Gestures in Compose 48.1 Swipe gestures and anchors 48.2 Detecting swipe gestures 48.3 Declaring the anchors map 48.4 Declaring thresholds 48.5 Moving a component in response to a swipe 48.6 About the SwipeDemo project 48.7 Creating the SwipeDemo project 48.8 Setting up the swipeable state and anchors 48.0 Designing the parent Per		427
48.	Detecting Swipe Gestures in Compose 48.1 Swipe gestures and anchors 48.2 Detecting swipe gestures 48.3 Declaring the anchors map 48.4 Declaring thresholds 48.5 Moving a component in response to a swipe 48.6 About the SwipeDemo project 48.7 Creating the SwipeDemo project 48.8 Setting up the swipeable state and anchors 48.9 Designing the parent Box 48.10 Testing the main		427
48.	Detecting Swipe Gestures in Compose 48.1 Swipe gestures and anchors 48.2 Detecting swipe gestures 48.3 Declaring the anchors map 48.4 Declaring thresholds 48.5 Moving a component in response to a swipe 48.6 About the SwipeDemo project 48.7 Creating the SwipeDemo project 48.8 Setting up the swipeable state and anchors 48.9 Designing the parent Box. 48.10 Testing the project	427 427 428 428 428 428 429 429 429 429 429 429 429 423	427
48.	Detecting Swipe Gestures in Compose 48.1 Swipe gestures and anchors 48.2 Detecting swipe gestures 48.3 Declaring the anchors map 48.4 Declaring thresholds 48.5 Moving a component in response to a swipe 48.6 About the SwipeDemo project 48.7 Creating the SwipeDemo project 48.8 Setting up the swipeable state and anchors. 48.9 Designing the parent Box. 48.11 Summary.	427 427 428 428 428 428 429 429 429 429 429 430 433	427
48.49.	Detecting Swipe Gestures in Compose 48.1 Swipe gestures and anchors 48.2 Detecting swipe gestures 48.3 Declaring the anchors map 48.4 Declaring thresholds 48.5 Moving a component in response to a swipe 48.6 About the SwipeDemo project 48.7 Creating the SwipeDemo project 48.8 Setting up the swipeable state and anchors. 48.9 Designing the parent Box. 48.10 Testing the project. 48.11 Summary.		427
48.49.	Detecting Swipe Gestures in Compose 48.1 Swipe gestures and anchors 48.2 Detecting swipe gestures 48.3 Declaring the anchors map 48.4 Declaring thresholds 48.5 Moving a component in response to a swipe 48.6 About the SwipeDemo project 48.7 Creating the SwipeDemo project 48.8 Setting up the swipeable state and anchors. 48.9 Designing the parent Box. 48.10 Testing the project. 48.11 Summary 49.1 Material Design 2 vs Material Design 3		427
48.49.	Detecting Swipe Gestures in Compose 48.1 Swipe gestures and anchors 48.2 Detecting swipe gestures 48.3 Declaring the anchors map 48.4 Declaring thresholds 48.5 Moving a component in response to a swipe 48.6 About the SwipeDemo project 48.7 Creating the SwipeDemo project 48.8 Setting up the swipeable state and anchors. 48.9 Designing the parent Box. 48.10 Testing the project. 48.11 Summary 49.1 Material Design 2 vs Material Design 3 49.2 Material Design 2 Theming		427
48.49.	Detecting Swipe Gestures in Compose 48.1 Swipe gestures and anchors 48.2 Detecting swipe gestures 48.3 Declaring the anchors map 48.4 Declaring thresholds 48.5 Moving a component in response to a swipe 48.6 About the SwipeDemo project 48.7 Creating the SwipeDemo project 48.8 Setting up the swipeable state and anchors. 48.9 Designing the parent Box. 48.10 Testing the project. 48.11 Summary. Working with Compose Theming 49.1 Material Design 2 vs Material Design 3 49.3 Material Design 3 Theming		427
48.	Detecting Swipe Gestures in Compose 48.1 Swipe gestures and anchors 48.2 Detecting swipe gestures 48.3 Declaring the anchors map 48.4 Declaring thresholds 48.5 Moving a component in response to a swipe 48.6 About the SwipeDemo project 48.7 Creating the SwipeDemo project 48.8 Setting up the swipeable state and anchors. 48.10 Testing the project 48.11 Summary Working with Compose Theming 49.1 Material Design 2 vs Material Design 3 49.2 Material Design 3 Theming 49.3 Material Design 3 Theming		427
48.	Detecting Swipe Gestures in Compose48.1 Swipe gestures and anchors48.2 Detecting swipe gestures48.3 Declaring the anchors map48.4 Declaring thresholds48.5 Moving a component in response to a swipe48.6 About the SwipeDemo project48.7 Creating the SwipeDemo project48.8 Setting up the swipeable state and anchors48.9 Designing the parent Box48.10 Testing the project48.11 SummaryWorking with Compose Theming49.1 Material Design 2 vs Material Design 349.2 Material Design 2 Theming49.3 Material Design 3 Theming49.4 Building a Custom Theme.49.5 Summary		427
48.49.50.	Detecting Swipe Gestures in Compose48.1 Swipe gestures and anchors48.2 Detecting swipe gestures48.3 Declaring the anchors map48.4 Declaring thresholds48.5 Moving a component in response to a swipe48.6 About the SwipeDemo project48.7 Creating the SwipeDemo project48.8 Setting up the swipeable state and anchors48.9 Designing the parent Box48.10 Testing the project48.11 SummaryWorking with Compose Theming49.1 Material Design 2 vs Material Design 349.2 Material Design 3 Theming49.3 Material Design 3 Theming49.5 SummaryA Material Design 3 Theming Tutorial	427 427 428 428 428 428 429 429 429 429 430 433 433 435 435 435 438 439 440	427 435
48.49.50.	Detecting Swipe Gestures in Compose 48.1 Swipe gestures and anchors 48.2 Detecting swipe gestures 48.3 Declaring the anchors map 48.4 Declaring thresholds 48.5 Moving a component in response to a swipe 48.6 About the SwipeDemo project 48.7 Creating the SwipeDemo project 48.8 Setting up the swipeable state and anchors 48.10 Testing the project 48.11 Summary Working with Compose Theming 49.1 Material Design 2 vs Material Design 3 49.2 Material Design 3 Theming 49.5 Summary 49.6 Summary		427 435 441
48.49.50.	Detecting Swipe Gestures in Compose 48.1 Swipe gestures and anchors 48.2 Detecting swipe gestures 48.3 Declaring the anchors map 48.4 Declaring thresholds 48.5 Moving a component in response to a swipe 48.6 About the SwipeDemo project 48.7 Creating the SwipeDemo project 48.8 Setting up the swipeable state and anchors 48.9 Designing the parent Box 48.10 Testing the project 48.11 Summary Working with Compose Theming 49.1 Material Design 2 vs Material Design 3 49.2 Material Design 3 Theming 49.3 Material Design 3 Theming 49.5 Summary A Material Design 3 Theming Tutorial 50.1 Creating the ThemeDemo project 50.2 Adding the Material Design 3 library		427 435 441
48.49.50.	Detecting Swipe Gestures in Compose 48.1 Swipe gestures and anchors 48.2 Detecting swipe gestures 48.3 Declaring the anchors map 48.4 Declaring thresholds 48.5 Moving a component in response to a swipe 48.6 About the SwipeDemo project 48.7 Creating the SwipeDemo project 48.8 Setting up the swipeable state and anchors 48.9 Designing the parent Box 48.10 Testing the project 48.11 Summary Working with Compose Theming 49.1 Material Design 2 vs Material Design 3 49.2 Material Design 3 Theming 49.3 Material Design 3 Theming 49.4 Building a Custom Theme 49.5 Summary A Material Design 3 Theming Tutorial 50.1 Creating the ThemeDemo project 50.2 Adding the Material Design 3 library 50.3 Designing the user interface		427 435

Table of Contents

50.5 Adding the theme to the project	
50.6 Enabling dynamic colors	
50.7 Summary	
51. Creating, Testing, and Uploading an Android App Bundle	
51.1 The release preparation process	
51.2 Android app bundles	
51.3 Register for a Google Play Developer Console account	
51.4 Configuring the app in the console	
51.5 Enabling Google Play app signing	
51.6 Creating a keystore file	
51.7 Creating the Android app bundle	
51.8 Generating test APK files	
51.9 Uploading the app bundle to the Google Play Developer Console	
51.10 Exploring the app bundle	
51.11 Managing testers	
51.12 Rolling the app out for testing	
51.13 Uploading new app bundle revisions	
51.14 Analyzing the app bundle file	
51.15 Summary	
51.15 Summary	
51.15 Summary52. An Overview of Gradle in Android Studio	
 51.15 Summary	
 51.15 Summary 52. An Overview of Gradle in Android Studio	
 51.15 Summary	
 51.15 Summary	

Chapter 1

1. Start Here

The goal of this book is to teach you how to build Android applications using Jetpack Compose, Android Studio, and the Kotlin programming language.

Beginning with the basics, this book explains how to set up an Android Studio development environment.

The book also includes in-depth chapters introducing the Kotlin programming language including data types, operators, control flow, functions, lambdas, and object-oriented programming.

An introduction to the key concepts of Jetpack Compose and Android project architecture is followed by a guided tour of Android Studio in Compose development mode. The book also covers the creation of custom Composables and explains how these functions are combined to create user interface layouts including the use of row, column, box, and list components.

Other topics covered include data handling using state properties, key user interface design concepts such as modifiers, navigation bars, and user interface navigation. Additional chapters explore building your own re-usable custom layout components.

The book also includes chapters covering graphics drawing, user interface animation, transitions, and gesture handling.

Chapters are also included covering view models, SQLite databases, Room database access, the Database Inspector, live data, and custom theme creation.

Finally, the book explains how to package up a completed app and upload it to the Google Play Store for publication.

Along the way, the topics covered in the book are put into practice through detailed tutorials, the source code for which is also available for download.

Assuming you already have some rudimentary programming experience, are ready to download Android Studio and the Android SDK, and have access to a Windows, Mac, or Linux system, you are ready to get started.

1.1 For Kotlin programmers

This book has been designed to address the needs of both existing Kotlin programmers and those who are new to both Kotlin and Jetpack Compose app development. If you are familiar with the Kotlin programming language, you can probably skip the Kotlin specific chapters.

1.2 For new Kotlin programmers

If you are new to programming in Kotlin then the entire book is appropriate for you. Just start at the beginning and keep going.

1.3 Downloading the code samples

The source code and Android Studio project files for the examples contained in this book are available for download at:

https://www.ebookfrenzy.com/retail/compose/index.php

Start Here

The steps to load a project from the code samples into Android Studio are as follows:

- 1. From the Welcome to Android Studio dialog, click on the Open button option.
- 2. In the project selection dialog, navigate to and select the folder containing the project to be imported and click on OK.

1.4 Feedback

We want you to be satisfied with your purchase of this book. If you find any errors in the book or have any comments, questions or concerns please contact us at *feedback@ebookfrenzy.com*.

1.5 Errata

While we make every effort to ensure the accuracy of the content of this book, inevitably, a book covering a subject area of this size and complexity may include some errors and oversights. Any known issues with the book will be outlined, together with solutions at the following URL:

https://www.ebookfrenzy.com/errata/compose.html

If you find an error not listed in the errata, please let us know by emailing our technical support team at *feedback@ ebookfrenzy.com*.

Chapter 2

2. Setting up an Android Studio Development Environment

Before any work can begin on the development of an Android application, the first step is to configure a computer system to act as the development platform. This involves several steps consisting of installing the Android Studio Integrated Development Environment (IDE) which also includes the Android Software Development Kit (SDK), the Kotlin plug-in and OpenJDK Java development environment.

This chapter will cover the steps necessary to install the requisite components for Android application development on Windows, macOS, and Linux-based systems.

2.1 System requirements

Android application development may be performed on any of the following system types:

- Windows 8/10/11 64-bit
- macOS 10.14 or later running on Intel or Apple silicon
- Chrome OS device with Intel i5 or higher
- Linux systems with version 2.31 or later of the GNU C Library (glibc)
- Minimum of 8GB of RAM (see below)
- Approximately 8GB of available disk space
- 1280 x 800 minimum screen resolution

Although Android Studio will run on computers with 8GB of RAM, performance will be greatly improved on systems containing more memory. This is particularly an issue if you plan to test your apps using the Android Virtual Device emulator (AVD).

2.2 Downloading the Android Studio package

Most of the work involved in developing applications for Android will be performed using the Android Studio environment. The content and examples in this book were created based on Android Studio Bumble Bee 2021.1.1 using the Android API 32 SDK which, at the time of writing, are the current versions.

Android Studio is, however, subject to frequent updates so a newer version may have been released since this book was published.

The latest release of Android Studio may be downloaded from the primary download page which can be found at the following URL:

https://developer.android.com/studio/index.html

If this page provides instructions for downloading a newer version of Android Studio it is important to note that there may be some minor differences between this book and the software. A web search for "Android Studio Bumble Bee" should provide the option to download the older version if these differences become a problem.

Setting up an Android Studio Development Environment

Alternatively, visit the following web page to find Android Studio Bumble Bee 2021.1.1 in the archives:

https://developer.android.com/studio/archive

2.3 Installing Android Studio

Once downloaded, the exact steps to install Android Studio differ depending on the operating system on which the installation is being performed.

2.3.1 Installation on Windows

Locate the downloaded Android Studio installation executable file (named *android-studio-<version>-windows*. *exe*) in a Windows Explorer window and double-click on it to start the installation process, clicking the Yes button in the User Account Control dialog if it appears.

Once the Android Studio setup wizard appears, work through the various screens to configure the installation to meet your requirements in terms of the file system location into which Android Studio should be installed and whether or not it should be made available to other users of the system. When prompted to select the components to install, make sure that the *Android Studio* and *Android Virtual Device* options are all selected.

Although there are no strict rules on where Android Studio should be installed on the system, the remainder of this book will assume that the installation was performed into *C*:*Program Files**Android**Android Studio* and that the Android SDK packages have been installed into the user's *AppData**Local**Android**sdk* sub-folder. Once the options have been configured, click on the *Install* button to begin the installation process.

On versions of Windows with a Start menu, the newly installed Android Studio can be launched from the entry added to that menu during the installation. The executable may be pinned to the taskbar for easy access by navigating to the *Android Studio\bin* directory, right-clicking on the *studio64* executable, and selecting the *Pin to Taskbar* menu option (on Windows 11 this option can be found by selecting *Show more options* from the menu).

2.3.2 Installation on macOS

Android Studio for macOS is downloaded in the form of a disk image (.dmg) file. Once the *android-studio-<version>-mac.dmg* file has been downloaded, locate it in a Finder window and double-click on it to open it as shown in Figure 2-1:





To install the package, simply drag the Android Studio icon and drop it onto the Applications folder. The Android Studio package will then be installed into the Applications folder of the system, a process that will typically take a few seconds to complete.

To launch Android Studio, locate the executable in the Applications folder using a Finder window and doubleclick on it. For future, easier access to the tool, drag the Android Studio icon from the Finder window and drop it onto the dock.

2.3.3 Installation on Linux

Having downloaded the Linux Android Studio package, open a terminal window, change directory to the location where Android Studio is to be installed and execute the following command:

unzip /<path to package>/android-studio-ide-<version>-linux.zip

Note that the Android Studio bundle will be installed into a subdirectory named *android-studio*. Assuming, therefore, that the above command was executed in */home/demo*, the software packages will be unpacked into */ home/demo/android-studio*.

To launch Android Studio, open a terminal window, change directory to the *android-studio/bin* sub-directory and execute the following command:

./studio.sh

When running on a 64-bit Linux system, it may be necessary to install some 32-bit support libraries before Android Studio will run. On Ubuntu these libraries can be installed using the following command:

```
sudo apt-get install libc6:i386 libncurses5:i386 libstdc++6:i386 lib32z1 libbz2-
1.0:i386
```

On Red Hat and Fedora-based 64-bit systems, use the following command:

sudo yum install zlib.i686 ncurses-libs.i686 bzip2-libs.i686

2.4 The Android Studio setup wizard

If you are installing Android Studio for the first time the initial dialog that appears once the setup process starts may resemble that shown in Figure 2-2 below:



Figure 2-2

If this dialog appears, click the Next button to display the SDK Components Setup dialog (Figure 2-3). Within this dialog, make sure that the Android SDK option is selected along with the latest API package before clicking on the Next button:

Setting up an Android Studio Development Environment

Android Stud	io Setup Wizard
SDK Components Setup	
Check the components you want to update/install. Cli	ck Next to continue.
Android SDK - (155 MB)	
Android SDK Platform	The collection of Android platform APIs, tools and utilities that enables you to debug, profile,
API 31: Android 12 Preview (S) – (104 MB)	and compile your apps. The setup wizard will update your current
Android SDK Location:	Total download size: 259 MB
/Users/neilsmyth/Library/Android/sdk	135 GB
	Cancel Previous Next Finish

Figure 2-3

After clicking Next, Android Studio will download and install the Android SDK and tools.

If you have previously installed an earlier version of Android Studio, the first time that this new version is launched, a dialog may appear providing the option to import settings from a previous Android Studio version. If you have settings from a previous version and would like to import them into the latest installation, select the appropriate option and location. Alternatively, indicate that you do not need to import any previous settings and click on the OK button to proceed.

After these initial setup steps have been taken, click the Finish button to display the Welcome to Android Studio screen:



Figure 2-4

2.5 Installing additional Android SDK packages

The steps performed so far have installed the Android Studio IDE and the current set of default Android SDK packages. Before proceeding, it is worth taking some time to verify which packages are installed and to install any missing or updated packages.

This task can be performed by clicking on the *More Actions* link within the welcome dialog and selecting the *SDK Manager* option from the drop-down menu. Once invoked, the *Android SDK* screen of the Preferences dialog will appear as shown in Figure 2-5:

• • •	Preferences				
Q	Appearance & Behavior \rightarrow System Settings \rightarrow A	ndroid SDK			
Appearance & Behavior	Manager for the Android SDK and Tools used by the	e IDE			
Appearance	Android SDK Location: /Users/neilsmyth/Library/A	ndroid/sdk		Edit Optimize disk space	
Menus and Toolbars	SDK Platforms SDK Tools SDK Update Sites	SDK Distforms SDK Tools SDK Lindate Sites			
✓ System Settings					
HTTP Proxy	Each Android SDK Platform package includes the an API level by default. Once installed the IDE will	Android platform a	and sources per	taining to	
Data Sharing	"show package details" to display individual SDK c	omponents.	eck for updates.	CHECK	
Date Formats	Name	API Level	Revision	Status	
Updates	Android API 32	32	1	Installed	
Brooms Elevation	Android 12.0 (S)	31	1	Partially installed	
Process Lievation	Android 11.0 (R)	30	3	Not installed	
Passwords	Android 10.0 (Q)	29	5	Not installed	
Android SDK	Android 9.0 (Pie)	28	6	Not installed	
Memory Settings	Android 8.1 (Oreo)	27	3	Not installed	
Notifications	Android 8.0 (Oreo)	26	2	Not installed	
Quick Lists	Android 7.1.1 (Nougat)	25	3	Not installed	
Path Variables	 Android 7.0 (Nougat) 	24	2	Not installed	
run vanabies	Android 6.0 (Marshmallow)	23	3	Not installed	
кеутар	Android 5.1 (Lollipop)	22	2	Not installed	
> Editor	Android 5.0 (Lollipop)	21	2	Not installed	
> Build, Execution, Deployment	Android 4.4W (KitKat Wear)	20	2	Not installed	
Kotlin	Android 4.4 (KitKat)	19	4	Not installed	
Tools	Android 4.3 (Jelly Bean)	18	3	Not installed	
/ 10013	Android 4.2 (Jelly Bean)	17	3	Not installed	
	Android 4.1 (Jelly Bean)	16	5	Not installed	
	Android 4.0.3 (IceCreamSandwich)	15	5	Not installed	
	 Android 4.0 (IceCreamSandwich) 	14	4	Not installed	
	Android 3.2 (Honevcomb)	13	1	Not installed	
		Hide	Obsolete Packag	ges 🗌 Show Package Details	

Figure 2-5

Immediately after installing Android Studio for the first, time it is likely that only the latest released version of the Android SDK has been installed. To install older versions of the Android SDK simply select the checkboxes corresponding to the versions and click on the *Apply* button.

Most of the examples in this book will support older versions of Android as far back as Android 8.0 (Oreo). This is to ensure that the apps run on a wide range of Android devices. Within the list of SDK versions, enable the checkbox next to Android 8.0 (Oreo) and click on the Apply button. In the resulting confirmation dialog click on the OK button to install the SDK. Subsequent dialogs will seek the acceptance of licenses and terms before performing the installation. Click Finish once the installation is complete.

It is also possible that updates will be listed as being available for the latest SDK. To access detailed information about the packages that are ready to be updated, enable the *Show Package Details* option located in the lower right-hand corner of the screen. This will display information similar to that shown in Figure 2-6:

Name	API Level	Revision	Status
Android TV Intel x86 Atom System Image	25	6	Not installed
Android Wear for China ARM EABI v7a System Image	25	3	Not installed
Android Wear for China Intel x86 Atom System Image	25	3	Not installed
Android Wear ARM EABI v7a System Image	25	3	Not installed
Android Wear Intel x86 Atom System Image	25	3	Not installed
Google APIs ARM 64 v8a System Image	25	8	Not installed
Google APIs ARM EABI v7a System Image	25	8	Not installed
Google APIs Intel x86 Atom System Image	25	8	Not installed
Google APIs Intel x86 Atom_64 System Image	25	6 🧲	Update Available: 8
Android 7.0 (Nougat)			
Google APIs	24	1	Not installed

Figure 2-6

The above figure highlights the availability of an update. To install the updates, enable the checkbox to the left of the item name and click on the *Apply* button.

In addition to the Android SDK packages, several tools are also installed for building Android applications. To view the currently installed packages and check for updates, remain within the SDK settings screen and select the SDK Tools tab as shown in Figure 2-7:

Setting up an Android Studio Development Environment

•	Manager for the Android SDK and Tools used by the IDE		
Appearance & Benavior	Android SDK Location: /Users/neilsmyth/Library/Android/sdk		Edit Optimize disk space
Appearance			
Menus and Toolbars	SDK Platform SDK Tools SDK Update Sites		
 System Settings 	Below are the available SDK developer tools. Once installed, the	IDE will automatically o	back
HTTP Proxy	for updates. Check "show package details" to display available v	versions of an SDK Tool	
Data Sharing	Name	Version	Status
Date Formats	Android SDK Build-Tools 31		Installed
Updates	NDK (Side by side)		Not Installed
Process Elevation	Android SDK Command-line Tools (latest)		Not Installed
Passwords	CMake		Not Installed
Android SDK	Android Auto API Simulators	1	Not installed
Microid SDK	Android Auto Desktop Head Unit Emulator	1.1	Not installed
Memory Settings	Android Emulator	30.7.4	Installed
Notifications	Android SDK Platform-Tools	31.0.3	Installed
Quick Lists	Google Play APK Expansion library	1	Not installed
Path Variables	Google Play Instant Development SDK	1.9.0	Not installed

Figure 2-7

Within the Android SDK Tools screen, make sure that the following packages are listed as *Installed* in the Status column:

- Android SDK Build-tools
- Android Emulator
- Android SDK Platform-tools
- Google Play Services
- Intel x86 Emulator Accelerator (HAXM installer)
- Google USB Driver (Windows only)
- Layout Inspector image server for API S

Note the Intel x86 Emulator Accelerator (HAXM installer) cannot be installed on Apple silicon-based Macs.

If any of the above packages are listed as *Not Installed* or requiring an update, simply select the checkboxes next to those packages and click on the *Apply* button to initiate the installation process. If the HAXM emulator settings dialog appears, select the recommended memory allocation:

• • •		HAXM			ر
Emulator Se	ettings				
We have detected that your Set the maximum amount of change these settings at an Refer to the Intel® HAXM Dr	system can run the Android emulator in RAM available for the Intel® Hardware y time by running the Intel® HAXM insta occumentation > for more information.	an accelerated perfor Accelerated Execution Iller.	mance mode. Manager (HAXM) to	use for all x86 emulato	r instances. You can
512.0 MB	2.0 GB (Recommended)	3.3 GB		4.6 GB	6.0 GB
			RAM allocation:	2,048 🖨 MiB	Use recommended size
				Cancel Previou	rs Next Finish

Figure 2-8

Once the installation is complete, review the package list and make sure that the selected packages are now listed as *Installed* in the *Status* column. If any are listed as *Not installed*, make sure they are selected and click on the *Apply* button again.

2.6 Making the Android SDK tools command-line accessible

Most of the time, the underlying tools of the Android SDK will be accessed from within the Android Studio environment. That being said, however, there will also be instances where it will be useful to be able to invoke those tools from a command prompt or terminal window. For the operating system on which you are developing to be able to find these tools, it will be necessary to add them to the system's *PATH* environment variable.

Regardless of operating system, the PATH variable needs to be configured to include the following paths (where <*path_to_android_sdk_installation>* represents the file system location into which the Android SDK was installed):

```
<path_to_android_sdk_installation>/sdk/tools
<path_to_android_sdk_installation>/sdk/tools/bin
<path_to_android_sdk_installation>/sdk/platform-tools
```

The location of the SDK on your system can be identified by launching the SDK Manager and referring to the *Android SDK Location:* field located at the top of the settings panel as highlighted in Figure 2-9:

Appearance & Behavior $ ightarrow$ System Settings $ ightarrow$ Android SDK		
Manager for the Android SDK and Tools used by the IDE		
Android SDK Location /Users/neilsmyth/Library/Android/sdk	Edit	Optimize disk space
SDK Platforms SDK Tools SDK Update Sites		
Each Android SDK Platform package includes the Android platform and sources pertain an API level by default. Once installed, the IDE will automatically check for updates. Che "show package details" to display individual SDK components.	ing to eck	

Figure 2-9

Once the location of the SDK has been identified, the steps to add this to the PATH variable are operating system dependent:

2.6.1 Windows 8.1

- 1. On the start screen, move the mouse to the bottom right-hand corner of the screen and select Search from the resulting menu. In the search box, enter Control Panel. When the Control Panel icon appears in the results area, click on it to launch the tool on the desktop.
- 2. Within the Control Panel, use the Category menu to change the display to Large Icons. From the list of icons select the one labeled System.
- 3. In the Environment Variables dialog, locate the Path variable in the System variables list, select it and click on the *Edit...* button. Using the *New* button in the edit dialog, add three new entries to the path. For example, assuming the Android SDK was installed into C:\Users\demo\AppData\Local\Android\Sdk, the following entries would need to be added:

C:\Users\demo\AppData\Local\Android\Sdk\platform-tools

C:\Users\demo\AppData\Local\Android\Sdk\tools

C:\Users\demo\AppData\Local\Android\Sdk\tools\bin

4. Click on OK in each dialog box and close the system properties control panel.

Open a command prompt window by pressing Windows + R on the keyboard and entering *cmd* into the Run dialog. Within the Command Prompt window, enter:

echo %Path%

The returned path variable value should include the paths to the Android SDK platform tools folders. Verify that

Setting up an Android Studio Development Environment

the *platform-tools* value is correct by attempting to run the *adb* tool as follows:

adb

The tool should output a list of command-line options when executed.

Similarly, check the *tools* path setting by attempting to run the AVD Manager command-line tool (don't worry if the avdmanager tool reports a problem with Java - this will be addressed later):

avdmanager

If a message similar to the following message appears for one or both of the commands, it is most likely that an incorrect path was appended to the Path environment variable:

'adb' is not recognized as an internal or external command, operable program or batch file.

2.6.2 Windows 10

Right-click on the Start menu, select Settings from the resulting menu and enter "Edit the system environment variables" into the *Find a setting* text field. In the System Properties dialog, click the *Environment Variables*... button. Follow the steps outlined for Windows 8.1 starting from step 3.

2.6.3 Windows 11

Right-click on the Start icon located in the taskbar and select Settings from the resulting menu. When the Settings dialog appears, scroll down the list of categories and select the "About" option. In the About screen, select *Advanced system settings* from the Related links section. When the System Properties window appears, click on the *Environment Variables...* button. Follow the steps outlined for Windows 8.1 starting from step 3.

2.6.4 Linux

On Linux, this configuration can typically be achieved by adding a command to the *.bashrc* file in your home directory (specifics may differ depending on the particular Linux distribution in use). Assuming that the Android SDK bundle package was installed into */home/demo/Android/sdk*, the export line in the *.bashrc* file would read as follows:

```
export PATH=/home/demo/Android/sdk/platform-tools:/home/demo/Android/sdk/tools:/
home/demo/Android/sdk/tools/bin:/home/demo/android-studio/bin:$PATH
```

Note also that the above command adds the *android-studio/bin* directory to the PATH variable. This will enable the *studio.sh* script to be executed regardless of the current directory within a terminal window.

2.6.5 macOS

Several techniques may be employed to modify the \$PATH environment variable on macOS. Arguably the cleanest method is to add a new file in the */etc/paths.d* directory containing the paths to be added to \$PATH. Assuming an Android SDK installation location of */Users/demo/Library/Android/sdk*, the path may be configured by creating a new file named *android-sdk* in the */etc/paths.d* directory containing the following lines:

```
/Users/demo/Library/Android/sdk/tools
/Users/demo/Library/Android/sdk/tools/bin
/Users/demo/Library/Android/sdk/platform-tools
```

Note that since this is a system directory it will be necessary to use the *sudo* command when creating the file. For example:

```
sudo vi /etc/paths.d/android-sdk
```

2.7 Android Studio memory management

Android Studio is a large and complex software application that consists of many background processes. Although Android Studio has been criticized in the past for providing less than optimal performance, Google has made significant performance improvements in recent releases and continues to do so with each new version. These improvements include allowing the user to configure the amount of memory used by both the Android Studio IDE and the background processes used to build and run apps. This allows the software to take advantage of systems with larger amounts of RAM.

If you are running Android Studio on a system with sufficient unused RAM to increase these values (this feature is only available on 64-bit systems with 5GB or more of RAM) and find that Android Studio performance appears to be degraded it may be worth experimenting with these memory settings. Android Studio may also notify you that performance can be increased via a dialog similar to the one shown below:



Figure 2-10

To view and modify the current memory configuration, select the *File -> Settings...* (*Android Studio -> Preferences...* on macOS) menu option and, in the resulting dialog, select the *Memory Settings* option listed under *System Settings* in the left-hand navigation panel as illustrated in Figure 2-11 below.

When changing the memory allocation, be sure not to allocate more memory than necessary or than your system can spare without slowing down other processes.

۹.	Appearance & Behavior > System Settings > Memory Settings			
 Appearance & Behavior Appearance Manus and Toolbars 	Configure the maximum amount of RAM the OS should allocate for Android Studio processes, such as the core IDE or Gradle daemon. Similar to allocating too little memory, allocating too much memory might degrade performance.			
V System Settings	IDE Heap Size Settings			
HTTP Proxy				
Data Sharing	IDE max heap size:	2048 MB - current		
Date Formats		This is a global setting that applies to all projects you open using Android Studio. You need to restart the IDE before any changes to its been size take effect		
Updates				
Process Elevation				

Figure 2-11

2.8 Updating Android Studio and the SDK

From time to time new versions of Android Studio and the Android SDK are released. New versions of the SDK are installed using the Android SDK Manager. Android Studio will typically notify you when an update is ready to be installed.

To manually check for Android Studio updates, use the *Help -> Check for Updates...* menu option from the Android Studio main window (*Android Studio -> Check for Updates...* on macOS).

2.9 Summary

Before beginning the development of Android-based applications, the first step is to set up a suitable development environment. This consists of the Android SDKs and Android Studio IDE (which also includes the OpenJDK development environment). In this chapter, we have covered the steps necessary to install these packages on Windows, macOS, and Linux.

Chapter 3

3. A Compose Project Overview

Now that we have installed Android Studio, the next step is to create an Android app using Jetpack Compose. Although this project will make use of several Compose features, it is an intentionally simple example intended to provide an early demonstration of Compose in action and an initial success on which to build as you work through the remainder of the book. The project will also serve to verify that your Android Studio environment is correctly installed and configured.

This chapter will create a new project using the Android Studio Compose project template and explore both the basic structure of a Compose-based Android Studio project and some of the key areas of Android Studio. In the next chapter, we will use this project to create a simple Android app.

Both chapters will briefly explain key features of Compose as they are introduced within the project. If anything is unclear when you have completed the project, rest assured that all of the areas covered in the tutorial will be explored in greater detail in later chapters of the book.

3.1 About the project

The completed project will consist of two text components and a slider. When the slider is moved, the current value will be displayed on one of the text components, while the font size of the second text instance will adjust to match the current slider position. Once completed, the user interface for the app will appear as shown in Figure 3-1:



Figure 3-1

3.2 Creating the project

The first step in building an app is to create a new project within Android Studio. Begin, therefore, by launching Android Studio so that the "Welcome to Android Studio" screen appears as illustrated in Figure 3-2:

•••	Welcome to Android Studio
Android Studio Arctic Fox 2020.3.1	
Projects	
Customize	Welcome to Android Studio
Plugins	Create a new project to start from scratch
Learn Android Studio	Open existing project from disk or version control.
	+ 😕 P
	New Project Open Get from VCS
	More Actions ~
\$	

Figure 3-2

Once this window appears, Android Studio is ready for a new project to be created. To create the new project, click on the *New Project* button to display the first screen of the *New Project* wizard.

3.3 Creating an activity

The next step is to define the type of initial activity that is to be created for the application. The left-hand panel provides a list of platform categories from which the *Phone and Tablet* option must be selected. Although a range of different activity types is available when developing Android applications, only the *Empty Compose Activity* template provides a pre-configured project ready to work with Compose. Select this option before clicking on the *Next* button:



Figure 3-3

3.4 Defining the project and SDK settings

In the project configuration window (Figure 3-4), set the *Name* field to *ComposeDemo*. The application name is the name by which the application will be referenced and identified within Android Studio and is also the name that would be used if the completed application were to go on sale in the Google Play store:

• •		New Project		
	Empty Compos	e Activity		
	Create a new e	mpty activity with Jetpack Compose		
	Name	ComposeDemo		
	Package name	com.ebookfrenzy.composedemo		
	Save location	/Users/neilsmyth/OneDrive/Documents/Books/Arctic_Fox_Java/WORK/ComposeDemo	5	
	Language	Kotlin	Ŧ	
	Minimum SDK	API 26: Android 8.0 (Oreo)	Ŧ	
		Your app will run on approximately 60.8% of devices. Help me choose		
		Use legacy android.support libraries ⑦ Using legacy android.support libraries will prevent you from using the latest Play Services and Jetpack libraries		
		Cancel Previous Next	F	inish

Figure 3-4

The *Package name* is used to uniquely identify the application within the Google Play app store application ecosystem. Although this can be set to any string that uniquely identifies your app, it is traditionally based on the reversed URL of your domain name followed by the name of the application. For example, if your domain is *www.mycompany.com*, and the application has been named *ComposeDemo*, then the package name might be specified as follows:

com.mycompany.composedemo

If you do not have a domain name you can enter any other string into the Company Domain field, or you may use *example.com* for testing, though this will need to be changed before an application can be published:

com.example.composedemo

The *Save location* setting will default to a location in the folder named *AndroidStudioProjects* located in your home directory and may be changed by clicking on the folder icon to the right of the text field containing the current path setting.

Set the minimum SDK setting to API 26: Android 8.0 (Oreo). This is the minimum SDK that will be used in most of the projects created in this book unless a necessary feature is only available in a more recent version. The objective here is to build an app using the latest Android SDK, while also retaining compatibility with devices running older versions of Android (in this case as far back as Android 8.0). The text beneath the Minimum SDK setting will outline the percentage of Android devices currently in use on which the app will run. Click on the *Help me choose* link to see a full breakdown of the various Android versions still in use:

A Compose Project Overview

• • •		Android Platform/API	Version Distribution			
ANDROID PLATFORM VERSION	API LEVEL	CUMULATIVE	Oreo System Custom data store JobScheduler improvements Cached data User Interface Picture-in-Picture mode Improved Notifications Autofill tranework	Security & Privacy		
4.0 Ice Cream Sandwich				New permissions New account access and discovery		
4.1 Jelly Bean		99.8%		APIs Runtime & Tools		
4.2 Jelly Bean		99.2%		Updated Java language support Updated ICU4J Android Framework		
4.3 Jelly Bean		98.4%	Downloadable fonts Multi-display support	APIs		
4.4 KitKat		98.1%	Adaptive icons Media VolumoShonor			
5.0 Lollipop		94.1%	Audio focus enhancements Media metrics			
5.1 Lollipop		92.3%	MediaPlayer and MediaRecorder improvements			
6.0 Marshmallow		84.9%	Improved media file access Wireless & Connectivity Wi-Fi Aware			
7.0 Nougat		73.7%	WI-FI Aware Bluetooth updates Companion device pairing			
7.1 Nougat		66.2%				
8.0 Oreo		60.8%				
8.1 Oreo	27	53.5%				
		39.5%				
9.0 Pie	28					
10. Android 10		8.2%	https://developer.android.com/about/	versions/oreo/android=8.0		
		_		Cancel		

Figure 3-5

Since Compose only works with Kotlin, the *Language* menu is preset to *Kotlin* and cannot be changed. Click on the *Finish* button to create the project.

3.5 Previewing the example project

At this point, Android Studio should have created a minimal example application project and opened the main window.





The newly created project and references to associated files are listed in the *Project* tool window located on the left-hand side of the main project window. The Project tool window has several modes in which information can be displayed. By default, this panel should be in *Android* mode. This setting is controlled by the menu at the top of the panel as highlighted in Figure 3-7. If the panel is not currently in Android mode, use the menu to

switch mode:





The code for the main activity of the project (an activity corresponds to a single user interface screen or module within an Android app) is contained within the *MainActivity.kt* file located under *app -> java -> com.example. composedemo* within the Project tool window as indicated in Figure 3-8:





Double-click on this file to load it into the main code editor panel. The editor can be used in different modes when writing code, the most useful of which when working with Compose is Split mode. The current mode can be changed using the buttons marked A in Figure 3-9. Split mode displays the code editor (B) alongside the Preview panel (C) in which the current user interface design will appear:

A Compose Project Overview



Figure 3-9

To get us started, Android Studio has already added some code to the *MainActivity.kt* file to display a Text component configured to display a message which reads "Hello Android".

If the project has not yet been built, the Preview panel will display the message shown in Figure 3-10:





If you see this notification, click on the *Build & Refresh* link to rebuild the project. After the build is complete, the Preview panel should update to display the user interface defined by the code in the *MainActivity.kt* file:





3.6 Reviewing the main activity

Android applications are created by bringing together one or more elements known as *Activities*. An activity is a single, standalone module of application functionality that either correlates directly to a single user interface screen and its corresponding functionality, or acts as a container for a collection of related screens. An appointments application might, for example, contain an activity screen that displays appointments set up for the current day. The application might also utilize a second activity consisting of multiple screens where new

appointments may be entered by the user and existing appointments edited.

When we created the ComposeDemo project, Android Studio created a single initial activity for our app, named it MainActivity, and generated some code for it in the *MainActivity.kt* file. This activity contains the first screen that will be displayed when the app is run on a device. Before we modify the code for our requirements in the next chapter, it is worth taking some time to review the code currently contained within the *MainActivity.kt* file.

The file begins with the following line (keep in mind that this may be different if you used your own domain name instead of *com.example*):

```
package com.example.composedemo
```

This tells the build system that the classes and functions declared in this file belong to the *com.example*. *composedemo* package which we configured when we created the project.

Next are a series of *import* directives. The Android SDK is comprised of a vast collection of libraries that provide the foundation for building Android apps. If all of these libraries were included within an app the resulting app bundle would be too large to run efficiently on a mobile device. To avoid this problem an app only imports the libraries that it needs to be able to run:

```
import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.material.MaterialTheme
import androidx.compose.material.Surface
import androidx.compose.material.Text
.
```

·

Initially, the list of import directives will most likely be "folded" to save space. To unfold the list, click on the small "+" button indicated by the arrow in Figure 3-12 below:



Figure 3-12

The MainActivity class is then declared as a subclass of the Android ComponentActivity class:

```
class MainActivity : ComponentActivity() {
   .
   .
}
```

The MainActivity class implements a single method in the form of *onCreate()*. This is the first method that is called when an activity is launched by the Android runtime system and is an artifact of the way apps used to be developed before the introduction of Compose. The *onCreate()* method is used here to provide a bridge between the containing activity and the Compose-based user interfaces that are to appear within it:

```
override fun onCreate(savedInstanceState: Bundle?) {
   super.onCreate(savedInstanceState)
   setContent {
      ComposeDemoTheme {
   }
}
```

```
•
}
}
```

.

The method declares that the content of the activity's user interface will be provided by a composable function named *ComposeDemoTheme*. This composable function is declared in the *Theme.kt* file located under the *app* -> <*package name*> -> *ui.theme* folder in the Project tool window. This, along with the other files in the *ui.theme* folder defines the colors, fonts, and shapes to be used by the activity and provides a central location from which to customize the overall theme of the app's user interface.

The call to the ComposeDemoTheme composable function is configured to contain a Surface composable. Surface is a built-in Compose component designed to provide a background for other composables:

```
ComposeDemoTheme {
```

```
// A surface container using the 'background' color from the theme
Surface(
    modifier = Modifier.fillMaxSize(),
    color = MaterialTheme.colors.background
.
.
.
.
```

In this case, the Surface component is configured to fill the entire screen and with the background set to the standard background color defined by the Android Material Design theme. Material Design is a set of design guidelines developed by Google to provide a consistent look and feel across all Android apps. It includes a theme (including fonts and colors), a set of user interface components (such as button, text, and a range of text fields), icons, and generally defines how an Android app should look, behave and respond to user interactions.

Finally, the Surface is configured to contain a composable function named Greeting which is passed a string value that reads "Android":

```
ComposeDemoTheme {
    // A surface container using the 'background' color from the theme
    Surface(
        modifier = Modifier.fillMaxSize(),
        color = MaterialTheme.colors.background
    ) {
        Greeting("Android")
    }
}
```

Outside of the scope of the MainActivity class, we encounter our first composable function declaration within the activity. The function is named Greeting and is, unsurprisingly, marked as being composable by the @ *Composable* annotation:

```
@Composable
fun Greeting(name: String) {
    Text(text = "Hello $name!")
}
```

The function accepts a String parameter (labeled name) and calls the built-in Text composable, passing through

20
a string value containing the word "Hello" concatenated with the name parameter. As will soon become evident as you work through the book, composable functions are the fundamental building blocks for developing Android apps using Compose.

The second composable function declared in the MainActivity.kt file reads as follows:

```
@Preview(showBackground = true)
@Composable
fun DefaultPreview() {
    ComposeDemoTheme {
        Greeting("Android")
    }
}
```

Earlier in the chapter, we looked at how the Preview panel allows us to see how the user interface will appear without having to compile and run the app. At first glance, it would be easy to assume that the preview rendering is generated by the code in the *onCreate()* method. In fact, that method only gets called when the app runs on a device or emulator. Previews are generated by preview composable functions. The *@Preview* annotation associated with the function tells Android Studio that this is a preview function and that the content emitted by the function is to be displayed in the Preview panel. As we will see later in the book, a single activity can contain multiple preview composable functions configured to preview specific sections of a user interface using different data values.

In addition, each preview may be configured by passing parameters to the *@Preview* annotation. For example, to view the preview with the rest of the standard Android screen decorations, modify the preview annotation so that it reads as follows:

```
@Preview(showSystemUi = true)
```

Once the preview has been updated, it should now be rendered as shown in Figure 3-13:



Figure 3-13

A Compose Project Overview

3.7 Preview updates

One final point worth noting is that the Preview panel is live and will automatically reflect minor changes made to the composable functions that make up a preview. To see this in action, edit the call to the Greeting function in the *DefaultPreview()* preview composable function to change the name from "Android" to "Compose". Note that as you make the change in the code editor, it is reflected in the preview.

More significant changes will require a build and refresh before being reflected in the preview. When this is required, Android Studio will display the following notice at the top of the Preview panel:



Figure 3-14

Simply click on the Build & Refresh link to update the preview for the latest changes.

The Preview panel also includes an interactive mode that allows you to trigger events on the user interface components (for example clicking buttons, moving sliders, scrolling through lists, etc.). Since ComposeDemo contains only an inanimate Text component at this stage, it makes more sense to introduce interactive mode in the next chapter.

3.8 Summary

In this chapter, we have created a new project using Android Studio's *Empty Compose Activity* template and explored some of the code automatically generated for the project. We have also introduced several features of Android Studio designed to make app development with Compose easier. The most useful features, and the places where you will spend most of your time while developing Android apps, are the code editor and Preview panel.

While the default code in the *MainActivity.kt* file provides an interesting example of a basic user interface, it bears no resemblance to the app we want to create. In the next chapter, we will modify and extend the app by removing some of the template code and writing our own composable functions.

Chapter 4

4. An Example Compose Project

In the previous chapter, we created a new Compose-based Android Studio project named ComposeDemo and took some time to explore both Android Studio and some of the project code that it generated to get us started. With those basic steps covered, this chapter will use the ComposeDemo project as the basis for a new app. This will involve the creation of new composable functions, introduce the concept of state, and make use of the Preview panel in interactive mode. As with the preceding chapter, key concepts explained in basic terms here will be covered in significantly greater detail in later chapters.

4.1 Getting started

Start Android Studio if it is not already running and open the ComposeDemo project created in the previous chapter. Once the project has loaded, double-click on the *MainActivity.kt* file (located in Project tool window under *app -> java -> <package name>*) to open it in the code editor. If necessary, switch the editor into Split mode so that both the editor and Preview panel are visible.

4.2 Removing the template Code

Within the *MainActivity.kt* file, delete some of the template code so that the file reads as follows:

```
package com.example.composedemo
class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            ComposeDemoTheme {
                Surface(
                     modifier = Modifier.fillMaxSize(),
                     color = MaterialTheme.colors.background
                ) {
                     Greeting("Android")
                }
            }
        }
    }
}
@Composable
fun Greeting(name: String) {
    Text(text = "Hello $name!")
+
@Preview(showSystemUi = true)
```

```
An Example Compose Project

@Composable

fun DefaultPreview() {

ComposeDemoTheme {

Greeting("Android")

}
```

4.3 The Composable hierarchy

Before we start to write the composable functions that will make up our user interface, it helps to first visualize the relationships between these components. The ability of one composable to call other composables essentially allows us to build a hierarchy tree of components. Once completed, the composable hierarchy for our ComposeDemo main activity can be represented as shown in Figure 4-1:





All of the elements in the above diagram, except for ComponentActivity, are composable functions. Of those functions, the Surface, Column, Spacer, Text, and Slider functions are built-in composables provided by Compose. The DemoScreen, DemoText, and DemoSlider composables, on the other hand, are functions that we will create to provide both structure to the design and the custom functionality we require for our app. The ComposeDemoTheme composable declaration can be found in the *ui.theme -> Theme.kt* file.

4.4 Adding the DemoText composable

We are now going to add a new composable function to the activity to represent the DemoText item in the hierarchy tree. The purpose of this composable is to display a text string using a font size value which adjusts in real-time as the slider is moved. Place the cursor beneath the final closing brace (}) of the MainActivity declaration and add the following function declaration:

```
@Composable
fun DemoText() {
}
```

The @Composable annotation notifies the build system that this is a composable function. When the function is

called, the plan is for it to be passed both a text string and the font size at which that text is to be displayed. This means that we need to add some parameters to the function:

```
@Composable
fun DemoText(message: String, fontSize: Float) {
}
```

The next step is to make sure the text is displayed. To achieve this, we will make a call to the built-in Text composable, passing through as parameters the message string, font size and, to make the text more prominent, a bold font weight setting:

```
@Composable
fun DemoText(message: String, fontSize: Float) {
    Text(
        text = message,
        fontSize = fontSize.sp,
        fontWeight = FontWeight.Bold
    )
}
```

Note that after making these changes, the code editor is indicating that "sp" and "FontWeight" are undefined. This is happening because these are defined and implemented in libraries that have not yet been imported into the *MainActivity.kt* file. One way to resolve this is to click on an undefined declaration so that it highlights as shown below, and then press Alt+Enter (Opt+Enter on macOS) on the keyboard to automatically import the missing library:



Alternatively, the missing import statements may be added manually to the list at the top of the file:

```
.
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.unit.sp
.
```

In the remainder of this book, all code examples will include any required library import statements.

We have now finished writing our first composable function. Notice that, except for the font weight, all the other properties are passed to the function when it is called (a function that calls another function is generally referred to as the *caller*). This increases the flexibility, and therefore re-usability, of the DemoText composable and is a key goal to keep in mind when writing composable functions.

An Example Compose Project

4.5 Previewing the DemoText composable

At this point, the Preview panel will most likely be displaying a message which reads "No preview found". The reason for this is that our *MainActivity.kt* file does not contain any composable functions prefixed with the @ Preview annotation. Add a preview composable function for DemoText to the *MainActivity.kt* file as follows:

```
@Preview
@Composable
fun DemoTextPreview() {
    DemoText(message = "Welcome to Android", fontSize = 12f)
}
```

After adding the preview composable, the Preview panel should have detected the change and displayed the link to build and refresh the preview rendering. Click the link and wait for the rebuild to complete, at which point the DemoText composable should appear as shown in Figure 4-3:



Figure 4-3

Minor changes made to the code in the *MainActivity.kt* file such as changing values will be instantly reflected in the preview without the need to build and refresh. For example, change the "Welcome to Android" text literal to "Welcome to Compose" and note that the text in the Preview panel changes as you type. Similarly, increasing the font size literal will instantly change the size of the text in the preview. This feature is referred to as Live Edit and can be enabled and disabled using the menu button indicated in Figure 4-4:





4.6 Adding the DemoSlider composable

The DemoSlider composable is a little more complicated than DemoText. It will need to be passed a variable containing the current slider position and an event handler function or lambda to call when the slider is moved by the user so that the new position can be stored and passed to the two Text composables. With these requirements in mind, add the function as follows:

```
import androidx.compose.foundation.layout.*
import androidx.compose.material.Slider
```

```
import androidx.compose.ui.unit.dp
.
.
@Composable
fun DemoSlider(sliderPosition: Float, onPositionChange: (Float) -> Unit ) {
    Slider(
        modifier = Modifier.padding(10.dp),
        valueRange = 20f..40f,
        value = sliderPosition,
        onValueChange = { onPositionChange(it) }
    )
}
```

The DemoSlider declaration contains a single Slider composable which is, in turn, passed four parameters. The first is a Modifier instance configured to add padding space around the slider. Modifier is a Kotlin class built into Compose which allows a wide range of properties to be set on a composable within a single object. Modifiers can also be created and customized in one composable before being passed to other composables where they can be further modified before being applied.

The second value passed to the Slider is a range allowed for the slider value (in this case the slider is limited to values between 20 and 40).

The next parameter sets the value of the slider to the position passed through by the caller. This ensures that each time DemoSlider is recomposed it retains the last position value.

Finally, we set the *onValueChange* parameter of the Slider to call the function or lambda we will be passing to the DemoSlider composable when we call it later. Each time the slider position changes, the call will be made and passed the current value which we can access via the Kotlin *it* keyword. We can further simplify this by assigning just the event handler parameter name (*onPositionChange*) and leaving the compiler to handle the passing of the current value for us:

```
onValueChange = onPositionChange
```

4.7 Adding the DemoScreen composable

The next step in our project is to add the DemoScreen composable. This will contain a variable named *sliderPosition* in which to store the current slider position and the implementation of the *handlePositionChange* event handler to be passed to the DemoSlider. This lambda will be responsible for storing the current position in the *sliderPosition* variable each time it is called with an updated value. Finally, DemoScreen will contain a Column composable configured to display the DemoText, Spacer, DemoSlider and the second, as yet to be added, Text composable in a vertical arrangement.

Start by adding the DemoScreen function as follows:

```
.

import androidx.compose.runtime.*

.

@Composable

fun DemoScreen() {
```

An Example Compose Project

}

```
var sliderPosition by remember { mutableStateOf(20f) }
val handlePositionChange = { position : Float ->
     sliderPosition = position
}
```

The *sliderPosition* variable declaration requires some explanation. As we will learn later, the Compose system repeatedly and rapidly *recomposes* user interface layouts in response to data changes. The change of slider position will, therefore, cause DemoScreen to be recomposed along with all of the composables it calls. Consider if we had declared and initialized our *sliderPosition* variable as follows:

var sliderPosition = 20f

Suppose the user slides the slider to position 21. The *handlePositionChange* event handler is called and stores the new value in the *sliderPosition* variable as follows:

```
val handlePositionChange = { position : Float ->
    sliderPosition = position
}
```

The Compose runtime system detects this data change and recomposes the user interface, including a call to the *DemoScreen* function which will, in turn, reinitialize the *sliderposition* variable to 20 causing the previous value of 21 to be lost. Declaring the *sliderPosition* variable in this way informs Compose that the current value needs to be remembered during recompositions:

```
var sliderPosition by remember { mutableStateOf(20f) }
```

The only remaining work within the DemoScreen implementation is to add a Column containing the required composable functions:

```
.
.
import androidx.compose.ui.Alignment
.
.
@Composable
fun DemoScreen() {
   var sliderPosition by remember { mutableStateOf(20f) }
   val handlePositionChange = { position : Float ->
      sliderPosition = position
   }
   Column(
      horizontalAlignment = Alignment.CenterHorizontally,
      verticalArrangement = Arrangement.Center,
      modifier = Modifier.fillMaxSize()
   ) {
```

```
DemoText(message = "Welcome to Compose", fontSize = sliderPosition)
```

```
Spacer(modifier = Modifier.height(150.dp))
DemoSlider(
    sliderPosition = sliderPosition,
    onPositionChange = handlePositionChange
)
Text(
    style = MaterialTheme.typography.h2,
    text = sliderPosition.toInt().toString() + "sp"
)
```

Points to note regarding these changes may be summarized as follows:

}

- When DemoSlider is called, it is passed a reference to our handlePositionChange event handler as the onPositionChange parameter.
- The Column composable accepts parameters that customize layout behavior. In this case, we have configured the column to center its children both horizontally and vertically.
- A Modifier has been passed to the Spacer to place a 150dp vertical space between the DemoText and DemoSlider components.
- The second Text composable is configured to use the h2 (Heading 2) style of the Material theme. The *sliderPosition* value is converted from a Float to an integer so that only whole numbers are displayed and then converted to a string value before being displayed to the user.

4.8 Previewing the DemoScreen composable

To confirm that the DemoScreen layout meets our expectations, we need to add a preview composable to the file. Note that the original DemoTextPreview composable may also be removed at this point:

```
.
@Preview(showBackground = true, showSystemUi = true)
@Composable
fun Preview() {
    ComposeDemoTheme {
        DemoScreen()
    }
}
@Preview(showBackground = true, showSystemUi = true)
@Composable
fun DemoTextPreview() {
    SimpleDemoTheme {
        DemoText(message = "Welcome to Compose", 25f)
    }
}
```

An Example Compose Project

ϯ

Note that we have enabled the *showSystemUi* property of the preview so that we will experience how the app will look when running on an Android device.

After performing a preview rebuild and refresh, the user interface should appear as originally shown in Figure 3-1.

4.9 Testing in interactive mode

At this stage, we know that the user interface layout for our activity looks how we want it to, but we don't know if it will behave as intended. One option is to run the app on an emulator or physical device (topics which are covered in later chapters). A quicker option, however, is to switch the preview panel into interactive mode. This is achieved by clicking on the button indicated in Figure 4-5 below:





When clicked, there will be a short delay when interactive mode starts, after which it should be possible to move the slider and watch the two Text components update accordingly:



Figure 4-6

Click the stop button (marked A in Figure 4-7 below) to exit interactive mode. If it appears that the preview needs to be refreshed, simply click on the *Build Refresh* button (B):





4.10 Completing the project

The final step is to make sure that the DemoScreen composable is called from within the Surface function located in the *onCreate()* method of the MainActivity class. Locate this method and modify it as follows:

This will ensure that, in addition to appearing in the preview panel, our user interface will also be displayed when the app runs on a device or emulator (a topic that will be covered in later chapters).

4.11 Summary

In this chapter, we have extended our ComposeDemo project to include some additional user interface elements in the form of two Text composables, a Spacer, and a Slider. These components were arranged vertically using a Column composable. We also introduced the concept of mutable state variables and explained how they are used to ensure that the app remembers state when the Compose runtime performs recompositions. The example also demonstrated how to use event handlers to respond to user interaction (in this case the user moving a slider). Finally, we made use of the Preview panel in interactive mode to test the app without the need to compile and run it on an emulator or physical device.

Although the Android Studio Preview panel allows us to see the layout we are designing and test basic functionality using interactive mode, it be will necessary to compile and run an entire app to fully test that it works. An Android application may be tested by installing and running it either on a physical device or in an *Android Virtual Device (AVD)* emulator environment. Before an AVD can be used, it must first be created and configured to match the specifications of a particular device model. In this chapter, we will work through the creation of such a virtual device using the Pixel 4 phone as a reference example.

5.1 About Android Virtual Devices

AVDs are essentially emulators that allow Android applications to be tested without the necessity to install the application on a physical Android-based device. An AVD may be configured to emulate a variety of hardware features including options such as screen size, memory capacity, and the presence or otherwise of features such as a camera, GPS navigation support, or an accelerometer. As part of the standard Android Studio installation, several emulator templates are installed allowing AVDs to be configured for a range of different devices. Custom configurations may be created to match any physical Android device by specifying properties such as processor type, memory capacity, and the size and pixel density of the screen.

An AVD session can appear either as a separate window or embedded within the Android Studio window.

New AVDs are created and managed using the Android Virtual Device Manager, which may be used either in command-line mode or with a more user-friendly graphical user interface. To create a new AVD, the first step is to launch the AVD Manager. This can be achieved from within the Android Studio environment by selecting the *Tools -> Device Manager* menu option from within the main window.

Virtual	Physical			\$
Create	device ?			
Device 🔺		API	Size on Disk	Actions
	No virtual devices added. Create applications without owning a Create virtual de	a virtual de a physical c evice	evice to tes levice.	

Once launched, the manager will appear as a tool window as shown in Figure 5-1:

Figure 5-1

To add a new AVD, begin by making sure that the Virtual tab is selected before clicking on the Create device

button to open the Virtual Device Configuration dialog:

• • •			Virt	ual Devic	e Configu	ration
🚺 Se	lect Hard	ware				
Choose a dev	vice definitio	n				
	Q					D Pixel 4
Category	Name 🔻	Play Store	Size	Resolution	Density	
τv	Pixel XL		5.5"	144	560	1000
Phone	Pixel 5		6.0"	108	440	Size: large
Wear OS	Pixel 4a		5.8"	108	440	Density: 440dpi
Tablet	Pixel 4 XL		6.3"	144	560	5.7" 2280px
Automotive	Pixel 4	⊳	5.7"	108	440	
	Pixel 3a XL		6.0"	108	400	
	Pixel 3a		5.6"	108	440	
	Pixel 3 XL		6.3"	144	560	
New Hardware	Profile Im	port Hardware	Profiles		G	Clone Device
(?)						Cancel Previous Next Finish

Figure 5-2

Within the dialog, perform the following steps to create a Pixel 4 compatible emulator:

- 1. From the *Category* panel, select the *Phone* option to display the list of available Android phone AVD templates.
- 2. Select the *Pixel 4* device option and click *Next*.
- 3. On the System Image screen, select the latest version of Android. Note that if the system image has not yet been installed a *Download* link will be provided next to the Release Name. Click this link to download and install the system image before selecting it. If the image you need is not listed, click on the *x86 Images* (or *ARM images* if you are running a Mac with Apple Silicon) and *Other images* tabs to view alternative lists.
- 4. Click *Next* to proceed and enter a descriptive name (for example *Pixel 4 API 32*) into the name field or simply accept the default name.
- 5. Click *Finish* to create the AVD.
- 6. With the AVD created, the Device Manager may now be closed. If future modifications to the AVD are necessary, simply re-open the Device Manager, select the AVD from the list, and click on the pencil icon in the Actions column to edit the settings.

5.2 Starting the emulator

To perform a test run of the newly created AVD emulator, simply select the emulator from the Device Manager and click on the launch button (the triangle in the Actions column). The emulator will appear embedded into the main Android Studio window and begin the startup process. The amount of time it takes for the emulator to start will depend on the configuration of both the AVD and the system on which it is running:



Figure 5-3

To hide and show the emulator tool window, click on the Emulator tool window button (marked A above). Click on the "x" close button next to the tab (B) to exit the emulator. The emulator tool window can accommodate multiple emulator sessions, with each session represented by a tab. Figure 5-4, for example, shows a tool window with two emulator sessions:



Figure 5-4

To switch between sessions, simply click on the corresponding tab.

Although the emulator probably defaulted to appearing in portrait orientation, this and other default options can be changed. Within the Device Manager, select the new Pixel 4 entry and click on the pencil icon in the *Actions* column of the device row. In the configuration screen, locate the *Startup orientation* section and change the orientation setting. Exit and restart the emulator session to see this change take effect. More details on the emulator are covered in the next chapter (*"Using and Configuring the Android Studio AVD Emulator"*).

To save time in the next section of this chapter, leave the emulator running before proceeding.

5.3 Running the application in the AVD

With an AVD emulator configured, the example ComposeDemo application created in the earlier chapter now can be compiled and run. With the ComposeDemo project loaded into Android Studio, make sure that the

newly created Pixel 4 AVD is displayed in the device menu (marked A in Figure 5-5 below), then either click on the run button represented by a green triangle (B), select the *Run -> Run 'app'* menu option or use the Ctrl-R keyboard shortcut:





The device menu (A) may be used to select a different AVD instance or physical device as the run target, and also to run the app on multiple devices. The menu also provides access to the Device Manager as well as device connection configuration and troubleshooting options:





The app can also be run on the currently selected target by clicking on the icon in the editor gutter next to the preview composable declaration as indicated by the arrow in Figure 5-7:





Once the application is installed and running, the user interface layout defined by the MainScreen function will appear within the emulator:



Figure 5-8

If the activity does not automatically launch, check to see if the launch icon has appeared among the apps on the emulator. If it has, simply click on it to launch the application. Once the run process begins, the Run tool window will become available. The Run tool window will display diagnostic information as the application package is installed and launched. Figure 5-9 shows the Run tool window output from a successful application launch:



Figure 5-9

If problems are encountered during the launch process, the Run tool window will provide information that will hopefully help to isolate the cause of the problem.

Assuming that the application loads into the emulator and runs as expected, we have safely verified that the Android development environment is correctly installed and configured.

5.4 Running on multiple devices

The run menu shown in Figure 5-6 above includes an option to run the app on multiple emulators and devices in parallel. When selected, this option displays the dialog shown in Figure 5-10 providing a list of both the AVDs configured on the system and any attached physical devices. Enable the checkboxes next to the emulators or devices to be targeted before clicking on the Run button:

	•	Run on Multiple Devices
Avai	lable o	devices
	Туре	Device
	G,	Pixel 4 API 30
	Q	8 Foldable API 30
	Q.	Pixel 3a API 29
		Cancel Run

Figure 5-10

After the Run button is clicked, Android Studio will launch the app on the selected emulators and devices.

5.5 Stopping a running application

To stop a running application, simply click on the stop button located in the main toolbar as shown in Figure 5-11:





An app may also be terminated using the Run tool window. Begin by displaying the *Run* tool window using the window bar button that becomes available when the app is running. Once the Run tool window appears, click the stop button highlighted in Figure 5-12 below:





5.6 Supporting dark theme

Android 10 introduced the much-awaited dark theme, support for which is enabled by default in Android Studio Compose-based app projects. To test dark theme in the AVD emulator, open the Settings app within the running Android instance in the emulator. Within the Settings app, choose the *Display* category and enable the *Dark theme* option as shown in Figure 5-13 so that the screen background turns black:





With dark theme enabled, run the ComposeDemo app and note that it appears using a dark theme including a black background and a purple background color on the button as shown in Figure 5-14:



Figure 5-14

Return to the Settings app and turn off Dark theme mode before continuing.

5.7 Running the emulator in a separate window

So far in this chapter, we have only used the emulator as a tool window embedded within the main Android Studio window. To run the emulator in a separate window, select the *File -> Settings...* menu option (*Android Studio -> Preferences...* on macOS), navigate to *Tools -> Emulator* in the left-hand navigation panel of the preferences dialog, and disable the *Launch in a tool window* option:



Figure 5-15

With the option disabled, click the Apply button followed by OK to commit the change, then exit the current emulator session by clicking on the close button on the tab marked B in Figure 5-3 above.

Run the sample app once again, at which point the emulator will appear as a separate window as shown below:





The choice of standalone or tool window mode is a matter of personal preference. If you prefer the emulator running in a tool window, return to the settings screen and re-enable the *Launch in a tool window* option. Before committing to standalone mode, however, keep in mind that the emulator tool window may also be detached from the main Android Studio window by clicking on the settings button (represented by the gear icon) in the tool emulator toolbar and selecting the *View Mode -> Float* menu option:



Figure 5-17

5.8 Enabling the device frame

The emulator can be configured to appear with (Figure 5-14) or without the device frame (Figure 5-16). To change the setting, open the Device Manager, select the AVD from the list, and click on the pencil icon in the Actions column to edit the settings. In the settings screen, locate and change the Enable Device Frame option:

AVD Name Pixel	4 API 32	
Pixel 4	5.7 1080x2280 xxhdpi	Change
ě	Android API 32 arm64-v8a	Change
Startup orientation		
	Portrait Landscape	
Emulated Performance	Graphics: Automatic	V

Figure 5-18

5.9 AVD command-line creation

As previously discussed, in addition to the graphical user interface it is also possible to create a new AVD directly from the command-line. This is achieved using the *avdmanager* tool in conjunction with some command-line options. Once initiated, the tool will prompt for additional information before creating the new AVD.

The avdmanager tool requires access to the Java Runtime Environment (JRE) to run. If, when attempting to run avdmanager, an error message appears indicating that the 'java' command cannot be found, the command prompt or terminal window within which you are running the command can be configured to use the OpenJDK environment bundled with Android Studio. Begin by identifying the location of the OpenJDK JRE as follows:

- 1. Launch Android Studio and open the ComposeDemo project created earlier in the book.
- 2. Select the File -> Settings... menu option (Android Studio -> Preferences... on macOS).
- 3. Navigate to the Build, Execution, Deployment section and select the Gradle option listed under the Build Tools category.
- 4. Click on the Gradle JDK setting and make a note of the path for Android Studio default JDK:



Figure 5-19

On Windows, execute the following command within the command prompt window from which avdmanager is to be run (where *<path to jre>* is replaced by the path copied from the Project Structure dialog above):

set JAVA_HOME=<path to jre>

On macOS or Linux, execute the following command:

export JAVA_HOME="<path to jre>"

If you expect to use the avdmanager tool frequently, follow the environment variable steps for your operating

system outlined in the chapter entitled "Setting up an Android Studio Development Environment" to configure JAVA_HOME on a system-wide basis.

Assuming that the system has been configured such that the Android SDK *tools* directory is included in the PATH environment variable, a list of available targets for the new AVD may be obtained by issuing the following command in a terminal or command window:

avdmanager list targets

The resulting output from the above command will contain a list of Android SDK versions that are available on the system. For example:

The avdmanager tool also allows new AVD instances to be created from the command-line. For example, to create a new AVD named *myAVD* using the target ID for the Android API level 29 device using the x86 ABI, the following command may be used:

```
avdmanager create avd -n myAVD -k "system-images;android-29;google_apis_
playstore;x86"
```

The avdmanager tool will create the new AVD to the specifications required for a basic Android 8 device, also providing the option to create a custom configuration to match the specification of a specific device if required. Once a new AVD has been created from the command-line, it may not show up in the Android Device Manager tool until the *Refresh* button is clicked.

In addition to the creation of new AVDs, several other tasks may be performed from the command-line. For example, a list of currently available AVDs may be obtained using the *list avd* command-line arguments:

```
avdmanager list avd
Available Android Virtual Devices:
   Name: Pixel_XL_API_28_No_Play
   Device: pixel_xl (Google)
    Path: /Users/neilsmyth/.android/avd/Pixel_XL_API_28_No_Play.avd
   Target: Google APIs (Google Inc.)
        Based on: Android API 28 Tag/ABI: google_apis/x86
   Skin: pixel_xl_silver
   Sdcard: 512M
```

Similarly, to delete an existing AVD, simply use the *delete* option as follows:

avdmanager delete avd -n <avd name>

5.10 Android Virtual Device configuration files

By default, the files associated with an AVD are stored in the *.android/avd* sub-directory of the user's home directory, the structure of which is as follows (where *<avd name>* is replaced by the name assigned to the AVD):

```
<avd name>.avd/config.ini
<avd name>.avd/userdata.img
<avd name>.ini
```

The *config.ini* file contains the device configuration settings such as display dimensions and memory specified during the AVD creation process. These settings may be changed directly within the configuration file and will be adopted by the AVD when it is next invoked.

The *<avd name>.ini* file contains a reference to the target Android SDK and the path to the AVD files. Note that a change to the *image.sysdir* value in the *config.ini* file will also need to be reflected in the *target* value of this file.

5.11 Moving and renaming an Android Virtual Device

The current name or the location of the AVD files may be altered from the command-line using the *avdmanager* tool's *move avd* argument. For example, to rename an AVD named Pixel4 to Pixel4a, the following command may be executed:

avdmanager move avd -n Pixel4 -r Pixel4a

To physically relocate the files associated with the AVD, the following command syntax should be used:

avdmanager move avd -n <avd name> -p <path to new location>

For example, to move an AVD from its current file system location to /tmp/Pixel4Test:

avdmanager move avd -n Pixel4 -p /tmp/Pixel4Test

Note that the destination directory must not already exist before executing the command to move an AVD.

5.12 Summary

A typical application development process follows a cycle of coding, compiling, and running in a test environment. Android applications may be tested on either a physical Android device or using an Android Virtual Device (AVD) emulator. AVDs are created and managed using the Android Studio Device Manager tool which may be used either as a command-line tool or via a graphical user interface. When creating an AVD to simulate a specific Android device model, the virtual device should be configured with a hardware specification matching that of the physical device.

The AVD emulator session may be displayed as a standalone window or embedded into the main Android Studio user interface.

Chapter 6

6. Using and Configuring the Android Studio AVD Emulator

Before the next chapter explores testing on physical Android devices, this chapter will take some time to provide an overview of the Android Studio AVD emulator and highlight many of the configuration features that are available to customize the environment in both standalone and tool window modes.

6.1 The emulator environment

When launched in standalone mode, the emulator displays an initial splash screen during the loading process. Once loaded, the main emulator window appears containing a representation of the chosen device type (in the case of Figure 6-1 this is a Pixel 4 device):



Figure 6-1

Positioned along the right-hand edge of the window is the toolbar providing quick access to the emulator controls and configuration options.

6.2 The emulator toolbar options

The emulator toolbar (Figure 6-2) provides access to a range of options relating to the appearance and behavior of the emulator environment.



Figure 6-2

Each button in the toolbar has associated with it a keyboard accelerator which can be identified either by hovering the mouse pointer over the button and waiting for the tooltip to appear or via the help option of the extended controls panel.

Though many of the options contained within the toolbar are self-explanatory, each option will be covered for the sake of completeness:

- Exit / Minimize The uppermost 'x' button in the toolbar exits the emulator session when selected while the '-' option minimizes the entire window.
- **Power** The Power button simulates the hardware power button on a physical Android device. Clicking and releasing this button will lock the device and turn off the screen. Clicking and holding this button will initiate the device "Power off" request sequence.
- Volume Up / Down Two buttons that control the audio volume of playback within the simulator environment.
- Rotate Left/Right Rotates the emulated device between portrait and landscape orientations.
- Take Screenshot Takes a screenshot of the content currently displayed on the device screen. The captured image is stored at the location specified in the Settings screen of the extended controls panel as outlined later in this chapter.
- Zoom Mode This button toggles in and out of zoom mode, details of which will be covered later in this chapter.
- Back Performs the standard Android "Back" navigation to return to a previous screen.
- Home Displays the device home screen.
- **Overview** Simulates selection of the standard Android "Overview" navigation which displays the currently running apps on the device.

- Fold Device Simulates the folding and unfolding of a foldable device. This option is only available if the emulator is running a foldable device system image.
- Extended Controls Displays the extended controls panel, allowing for the configuration of options such as simulated location and telephony activity, battery strength, cellular network type, and fingerprint identification.

6.3 Working in zoom mode

The zoom button located in the emulator toolbar switches in and out of zoom mode. When zoom mode is active the toolbar button is depressed and the mouse pointer appears as a magnifying glass when hovering over the device screen. Clicking the left mouse button will cause the display to zoom in relative to the selected point on the screen, with repeated clicking increasing the zoom level. Conversely, clicking the right mouse button decreases the zoom level. Toggling the zoom button off reverts the display to the default size.

Clicking and dragging while in zoom mode will define a rectangular area into which the view will zoom when the mouse button is released.

While in zoom mode the visible area of the screen may be panned using the horizontal and vertical scrollbars located within the emulator window.

6.4 Resizing the emulator window

The size of the emulator window (and the corresponding representation of the device) can be changed at any time by clicking and dragging on any of the corners or sides of the window.

6.5 Extended control options

The extended controls toolbar button displays the panel illustrated in Figure 6-3. By default, the location settings will be displayed. Selecting a different category from the left-hand panel will display the corresponding group of controls:

• •	•	Extended	l Controls - Pi	xel_4_API_3	1:5554		
0	Location						
⊒	Displays	GPS data point	Decimal	_	Latitude	Longitu	de
	Cellular	oboraniate bystem	Decima		37.422	-122.0	084
÷.	Battery	Currently reported locati	on		Altitude (meters)	Speed	(knots)
٨	Camera	Latitude: 37.4220 Longitude: -122.0840 Altitude: 5.0 Speed: 0.0			5.0	0.0	
r.	Phone						
0	Directional pad						Send
Ŷ	Microphone	GPS data playback					
Ť	Fingerprint	Delay (sec)	Latitude	Longitude	Elevation	Name	Description
ه)	Virtual sensors						
Ŭ	Bug report						
Ð	Snapshots						
	Record and Playback						
	Google Play						
\$	Settings						
?	Help	> Spe	eed 1X 🚽			Lo	ad GPX/KML

Figure 6-3

Using and Configuring the Android Studio AVD Emulator

6.5.1 Location

The location controls allow simulated location information to be sent to the emulator in the form of decimal or sexigesimal coordinates. Location information can take the form of a single location, or a sequence of points representing movement of the device, the latter being provided via a file in either GPS Exchange (GPX) or Keyhole Markup Language (KML) format. Alternatively, the integrated Google Maps panel may be used to visually select single points or travel routes.

6.5.2 Displays

In addition to the main display shown within the emulator screen, the Displays option allows additional displays to be added running within the same Android instance. This can be useful for testing apps for dual-screen devices such as the Microsoft Surface Duo. These additional screens can be configured to be any required size and appear within the same emulator window as the main screen.

6.5.3 Cellular

The type of cellular connection being simulated can be changed within the cellular settings screen. Options are available to simulate different network types (CSM, EDGE, HSDPA, etc) in addition to a range of voice and data scenarios such as roaming and denied access.

6.5.4 Battery

A variety of battery state and charging conditions can be simulated on this panel of the extended controls screen, including battery charge level, battery health, and whether the AC charger is currently connected.

6.5.5 Camera

The emulator simulates a 3D scene when the camera is active. This takes the form of the interior of a virtual building through which you can navigate by holding down the Option key (Alt on Windows) while using the mouse pointer and keyboard keys when recording video or before taking a photo within the emulator. This extended configuration option allows different images to be uploaded for display within the virtual environment.

6.5.6 Phone

The phone extended controls provide two very simple but useful simulations within the emulator. The first option allows for the simulation of an incoming call from a designated phone number. This can be of particular use when testing how an app handles high-level interrupts of this nature.

The second option allows the receipt of text messages to be simulated within the emulator session. As in the real world, these messages appear within the Message app and trigger the standard notifications within the emulator.

6.5.7 Directional pad

A directional pad (D-Pad) is an additional set of controls either built into an Android device or connected externally (such as a game controller) that provides directional controls (left, right, up, down). The directional pad settings allow D-Pad interaction to be simulated within the emulator.

6.5.8 Microphone

The microphone settings allow the microphone to be enabled and virtual headset and microphone connections to be simulated. A button is also provided to launch the Voice Assistant on the emulator.

6.5.9 Fingerprint

Many Android devices are now supplied with built-in fingerprint detection hardware. The AVD emulator makes it possible to test fingerprint authentication without the need to test apps on a physical device containing a fingerprint sensor. Details on how to configure fingerprint testing within the emulator will be covered in detail later in this chapter.

6.5.10 Virtual sensors

The virtual sensors option allows the accelerometer and magnetometer to be simulated to emulate the effects of the physical motion of a device such as rotation, movement, and tilting through yaw, pitch and roll settings.

6.5.11 Snapshots

Snapshots contain the state of the currently running AVD session to be saved and rapidly restored making it easy to return the emulator to an exact state. Snapshots are covered in later in this chapter.

6.5.12 Record and playback

Allows the emulator screen and audio to be recorded and saved in either WebM or animated GIF format.

6.5.13 Google Play

If the emulator is running a version of Android with Google Play Services installed, this option displays the current Google Play version and provides the option to update the emulator to the latest version.

6.5.14 Settings

The settings panel provides a small group of configuration options. Use this panel to choose a darker theme for the toolbar and extended controls panel, specify a file system location into which screenshots are to be saved, configure OpenGL support levels, and configure the emulator window to appear on top of other windows on the desktop.

6.5.15 Help

The Help screen contains three sub-panels containing a list of keyboard shortcuts, links to access the emulator online documentation, file bugs and send feedback, and emulator version information.

6.6 Working with snapshots

When an emulator starts for the very first time it performs a *cold boot* much like a physical Android device when it is powered on. This cold boot process can take some time to complete as the operating system loads and all the background processes are started. To avoid the necessity of going through this process every time the emulator is started, the system is configured to automatically save a snapshot (referred to as a *quick-boot snapshot*) of the emulator's current state each time it exits. The next time the emulator is launched, the quick-boot snapshot is loaded into memory and execution resumes from where it left off previously, allowing the emulator to restart in a fraction of the time needed for a cold boot to complete.

The Snapshots screen of the extended controls panel can be used to store additional snapshots at any point during the execution of the emulator. This saves the exact state of the entire emulator allowing the emulator to be restored to the exact point in time that the snapshot was taken. From within the screen, snapshots can be taken using the *Take Snapshot* button (marked A in Figure 6-4). To restore an existing snapshot, select it from the list (B) and click the run button (C) located at the bottom of the screen. Options are also provided to edit (D) the snapshot name and description and to delete (E) the currently selected snapshot:

Using and Configuring the Android Studio AVD Emulator

• •	0	Extended Controls - Pixel_4_API_31:5554	
0	Location	Snapshots Settings	
⊒	Displays	Quickboot	1001 0 0 VI
	Cellular	snap_2021-08-03_09-59-33	
7	Battery	Before app crash	
23	Camera		dollars
Ľ.	Phone		
0	Directional pad		Hello Work!
J	Microphone		
Ť	Fingerprint		CONVERT
(G).	Virtual sensors		
2 *	Dug report		
÷	Bug report		
Ð	Snapshots		Before app crash
	Record and Playback		778 MB,captured 8/3/21 9:59 AM File: snap_2021-08-03_09-59-53
	Google Play	000	
۵	Settings		
0	Help		TAKE SNAPSHOT

Figure 6-4

The Settings option (F) provides the option to configure the automatic saving of quick-boot snapshots (by default the emulator will ask whether to save the quick boot snapshot each time the emulator exits) and to reload the most recent snapshot. To force an emulator session to perform a cold boot instead of using a previous quick-boot snapshot, open the AVD Manager (*Tools -> AVD Manager*), click on the down arrow in the Actions column for the emulator and select the Cold Boot Now menu option.





6.7 Configuring fingerprint emulation

The emulator allows up to 10 simulated fingerprints to be configured and used to test fingerprint authentication within Android apps. To configure simulated fingerprints begin by launching the emulator, opening the Settings app, and selecting the *Security & Location* option.

Within the Security settings screen, select the *Use fingerprint* option. On the resulting information screen click on the *Next* button to proceed to the Fingerprint setup screen. Before fingerprint security can be enabled a backup screen unlocking method (such as a PIN) must be configured. Click on the *Fingerprint* + *PIN* button and, when prompted, choose not to require the PIN on device startup. Enter and confirm a suitable PIN and complete the PIN entry process by accepting the default notifications option.

Proceed through the remaining screens until the Settings app requests a fingerprint on the sensor. At this point display the extended controls dialog, select the *Fingerprint* category in the left-hand panel and make sure that

Finger 1 is selected in the main settings panel:

•	Location	Fingerprint Finger 1	~
	Cellular Battery		TOUCH THE SENSOR
L.	Phone		
0	Directional pad		
Ļ	Microphone		
Ċ	Fingerprint		



Click on the *Touch the Sensor* button to simulate Finger 1 touching the fingerprint sensor. The emulator will report the successful addition of the fingerprint:



Figure 6-7

To add additional fingerprints click on the *Add Another* button and select another finger from the extended controls panel menu before clicking on the *Touch the Sensor* button once again.

6.8 The emulator in tool window mode

As outlined in the previous chapter ("*Creating an Android Virtual Device (AVD) in Android Studio*"), Android Studio can be configured to launch the emulator as an embedded tool window so that it does not appear in a separate window. When running in this mode, the same controls available in standalone mode are provided in the toolbar as shown in Figure 6-8:

Using and Configuring the Android Studio AVD Emulator



Figure 6-8

From left to right, these buttons perform the following tasks (details of which match those for standalone mode):

- Power
- Volume Up
- Volume Down
- Rotate Left
- Rotate Right
- Back
- Home
- Overview
- Screenshot
- Snapshots
- Extended Controls

6.9 Summary

Android Studio contains an Android Virtual Device emulator environment designed to make it easier to test applications without the need to run on a physical Android device. This chapter has provided a brief tour of the emulator and highlighted key features that are available to configure and customize the environment to simulate different testing conditions.

Chapter 7

7. A Tour of the Android Studio User Interface

While it is tempting to plunge into running the example application created in the previous chapter, doing so involves using aspects of the Android Studio user interface which are best described in advance.

Android Studio is a powerful and feature-rich development environment that is, to a large extent, intuitive to use. That being said, taking the time now to gain familiarity with the layout and organization of the Android Studio user interface will considerably shorten the learning curve in later chapters of the book. With this in mind, this chapter will provide an initial overview of the various areas and components that make up the Android Studio environment.

7.1 The Welcome screen

The welcome screen (Figure 7-1) is displayed any time that Android Studio is running with no projects currently open (open projects can be closed at any time by selecting the *File -> Close Project* menu option). If Android Studio was previously exited while a project was still open, the tool will bypass the welcome screen next time it is launched, automatically opening the previously active project.



Figure 7-1

In addition to a list of recent projects, the welcome screen provides options for performing tasks such as opening and creating projects along with access to projects currently under version control. In addition, the *Customize* screen provides options to change the theme and font settings used by both the IDE and the editor. Android Studio plugins may be viewed, installed, and managed using the *Plugins* option.

Additional options are available by clicking on the menu button as shown in Figure 7-2:

A Tour of the Android Studio User Interface



Figure 7-2

7.2 The main window

When a new project is created, or an existing one opened, the Android Studio *main window* will appear. When multiple projects are open simultaneously, each will be assigned its own main window. The precise configuration of the window will vary depending on which tools and panels were displayed the last time the project was open, but will typically resemble that of Figure 7-3.



Figure 7-3

The various elements of the main window can be summarized as follows:

A - Menu Bar - Contains a range of menus for performing tasks within the Android Studio environment.

B – **Toolbar** – A selection of shortcuts to frequently performed actions. The toolbar buttons provide quick access to a select group of menu bar actions. The toolbar can be customized by right-clicking on the bar and selecting the *Customize Menus and Toolbars*... menu option. If the toolbar is not visible, it can be displayed using the *View* -> *Appearance* -> *Toolbar* menu option.

C – **Navigation Bar** – The navigation bar provides a convenient way to move around the files and folders that make up the project. Clicking on an element in the navigation bar will drop down a menu listing the sub-folders and files at that location ready for selection. Similarly, clicking on a class name displays a menu listing methods contained within that class. Select a method from the list to be taken to the corresponding location within the code editor. Hide and display this bar using the *View -> Appearance -> Navigation Bar* menu option.

D – **Editor Window** – The editor window displays the content of the file on which the developer is currently working. When multiple files are open, each file is represented by a tab located along the top edge of the editor as shown in Figure 7-4.

╉ MainActivity.kt 🛛	🛃 colors.xml 🛛 🗬 build.gradle (:app) 🛛	
		🗮 Code 🗉 Split 🗖 Design
26 📮	setContent {	×
27	ComposeDemoTheme {	

Figure 7-4

E - Status Bar – The status bar displays informational messages about the project and the activities of Android Studio together with the tools menu button located in the far left corner. Hovering over items in the status bar will display a description of that field. Many fields are interactive, allowing the user to click to perform tasks or obtain more detailed status information.

F – Project Tool Window – The project tool window provides a hierarchical overview of the project file structure allowing navigation to specific files and folders to be performed. The toolbar can be used to display the project in several different ways. The default setting is the *Android* view which is the mode primarily used in the remainder of this book.

The project tool window is just one of many tool windows available within the Android Studio environment.

7.3 The tool windows

In addition to the project view tool window, Android Studio also includes many other windows which, when enabled, are displayed along the bottom and sides of the main window. The tool window quick access menu can be displayed by hovering the mouse pointer over the button located in the far left-hand corner of the status bar (Figure 7-5) without clicking the mouse button.



Figure 7-5

Selecting an item from the quick access menu will cause the corresponding tool window to appear within the main window.

Alternatively, a set of *tool window bars* can be displayed by clicking on the quick access menu icon in the status bar. These bars appear along the left, right, and bottom edges of the main window (as indicated by the arrows in

A Tour of the Android Studio User Interface

Figure 7-6) and contain buttons for showing and hiding each of the tool windows. When the tool window bars are displayed, a second click on the button in the status bar will hide them.



Figure 7-6

Clicking on a button will display the corresponding tool window while a second click will hide the window. Buttons prefixed with a number (for example 1: Project) indicate that the tool window may also be displayed by pressing the Alt key on the keyboard (or the Command key for macOS) together with the corresponding number.

The location of a button in a tool window bar indicates the side of the window against which the window will appear when displayed. These positions can be changed by clicking and dragging the buttons to different locations in other window toolbars.

Each tool window has its own toolbar along the top edge. The buttons within these toolbars vary from one tool to the next, though all tool windows contain a settings option, represented by the cog icon, which allows various aspects of the window to be changed. Figure 7-7 shows the settings menu for the Project tool window. Options are available, for example, to undock a window and to allow it to float outside of the boundaries of the Android Studio main window, and to move and resize the tool panel.



Figure 7-7

All of the windows also include a far-right button on the toolbar providing an additional way to hide the tool window from view. A search of the items within a tool window can be performed simply by giving that window
focus by clicking in it and then typing the search term (for example the name of a file in the Project tool window). A search box will appear in the window's toolbar and items matching the search highlighted.

Android Studio offers a wide range of tool windows, the most commonly used of which are as follows:

- **App Inspector** Provides access to the Database and Background Task inspectors. The Database Inspector allows you to inspect, query, and modify your app's databases while the app is running. The Background Task Inspector allows background worker tasks created using WorkManager to be monitored and managed.
- **Build** The build tool window displays information about the build process while a project is being compiled and packaged and displays details of any errors encountered.
- **Build Variants** The build variants tool window provides a quick way to configure different build targets for the current application project (for example different builds for debugging and release versions of the application, or multiple builds to target different device categories).
- **Device File Explorer** Available via the *View -> Tool Windows -> Device File Explorer* menu, this tool window provides direct access to the filesystem of the currently connected Android device or emulator allowing the filesystem to be browsed and files copied to the local filesystem.
- **Device Manager** Provides access to the Device Manager tool window where physical Android device connections and emulators may be added, removed, and managed.
- **Emulator** Contains the AVD emulator if the option has been enabled to run the emulator in a tool window as outlined in the chapter entitled *"Creating an Android Virtual Device (AVD) in Android Studio"*.
- Event Log The event log window displays messages relating to events and activities performed within Android Studio. The successful build of a project, for example, or the fact that an application is now running will be reported within this tool window.
- **Favorites** A variety of project items can be added to the favorites list. Right-clicking on a file in the project view, for example, provides access to an *Add to Favorites* menu option. Similarly, a method in a source file can be added as a favorite by right-clicking on it in the Structure tool window. Anything added to a Favorites list can be accessed through this Favorites tool window.
- **Gradle** The Gradle tool window provides a view of the Gradle tasks that make up the project build configuration. The window lists the tasks that are involved in compiling the various elements of the project into an executable application. Right-click on a top-level Gradle task and select the *Open Gradle Config* menu option to load the Gradle build file for the current project into the editor. Gradle will be covered in greater detail later in this book.
- Layout Inspector Provides a visual 3D rendering of the hierarchy of components that make up a user interface layout.
- Logcat The Logcat tool window provides access to the monitoring log output from a running application in addition to options for taking screenshots and videos of the application and stopping and restarting a process.
- **Problems** A central location in which to view all of the current errors or warnings within the project. Double-clicking on an item in the problem list will take you to the problem file and location.
- **Profiler** The Android Profiler tool window provides real-time monitoring and analysis tools for identifying performance issues within running apps, including CPU, memory and network usage. This option becomes available when an app is currently running.
- Project The project view provides an overview of the file structure that makes up the project allowing for

A Tour of the Android Studio User Interface

quick navigation between files. Generally, double-clicking on a file in the project view will cause that file to be loaded into the appropriate editing tool.

- **Resource Manager** A tool for adding and managing resources and assets such as images, colors, and layout files contained with the project.
- **Run** The run tool window becomes available when an application is currently running and provides a view of the results of the run together with options to stop or restart a running process. If an application is failing to install and run on a device or emulator, this window will typically provide diagnostic information relating to the problem.
- **Structure** The structure tool provides a high-level view of the structure of the source file currently displayed in the editor. This information includes a list of items such as classes, methods, and variables in the file. Selecting an item from the structure list will take you to that location in the source file in the editor window.
- **Terminal** Provides access to a terminal window on the system on which Android Studio is running. On Windows systems, this is the Command Prompt interface, while on Linux and macOS systems this takes the form of a Terminal prompt.
- **TODO** As the name suggests, this tool provides a place to review items that have yet to be completed on the project. Android Studio compiles this list by scanning the source files that make up the project to look for comments that match specified TODO patterns. These patterns can be reviewed and changed by selecting the *File -> Settings...* menu option (*Android Studio -> Preferences...* on macOS) and navigating to the *TODO* page listed under *Editor*.

7.4 Android Studio keyboard shortcuts

Android Studio includes an abundance of keyboard shortcuts designed to save time when performing common tasks. A full keyboard shortcut keymap listing can be viewed and printed from within the Android Studio project window by selecting the *Help -> Keymap Reference* menu option. You may also list and modify the keyboard shortcuts by selecting the *File -> Settings...* menu option (*Android Studio -> Preferences...* on macOS) and clicking on the Keymap entry as shown in Figure 7-8 below:

• • •	Preferer	nces	
Qr	Keymap		
> Appearance & Behavior	macOS 🔻 🌣		
Keymap	Get more keymaps in Preferences Plu	igins	
> Editor			
Plugins	Ξ × 🖍 🛦	Q	
> Version Control	👻 🔄 Editor Actions		
> Build, Execution, Deployment	Add or Remove Caret		℃⇔Click
> Languages & Frameworks	Add Rectangular Selection or	n Mouse Drag	て合発Click
	Backspace		<u>a</u> 4a
~ 10015	Move Caret Backward a Parag	graph	
Web Browsers	Move Caret Backward a Parag	graph with Selection	
External Tools	Choose Lookup Item		4
Terminal @	Choose Lookup Item and Ins	ert Dot	
Device File Explorer	Close Cookup Item Replace	3	
> Diff & Merge	Clone Caret Below		
Emulator	Move Caret to Code Block Fr	nd	7.第1
Server Certificator	Move Caret to Code Block Er	nd with Selection	10#1
Server Certificates	Move Caret to Code Block St	tart	7.38[
Settings Repository	Move Caret to Code Block St	tart with Selection]\$\$67
Startup Tasks 👘	Complete Current Statement	t	\$\$\$\$
> Tasks 👘	Create Rectangular Selection	n	℃☆Middle-Click
> Other Settings	Create Rectangular Selection	n on Mouse Drag	Click Middle-Click
Experimental	Cut Line Backward		
	Cut up to Line End		^K
	Decrease Font Size		
	Delete		
	Delete Line		K Ø
	A Find Action, Move to Another Cha Assign custom shortcuts or change	ngelist shortcuts conflict with the macOS system settings.	ne macOS system shortcuts.
?			Cancel Apply OK

Figure 7-8

7.5 Switcher and recent files navigation

Another useful mechanism for navigating within the Android Studio main window involves the use of the *Switcher*. Accessed via the Ctrl-Tab keyboard shortcut, the switcher appears as a panel listing both the tool windows and currently open files (Figure 7-9).

Switcher	
≡ 0 Logcat	🛃 MainActivity.kt
1 Project	🗬 build.gradle (:app) 🔲
★ 2 Favorites	🖶 colors.xml
♂ 3 Profiler	
6 Problems	
7 Structure	
🖶 A App Inspection	
🔲 🛛 Device File Explorer	
🖫 E Emulator	
🗬 G Gradle	
🗟 I Layout Inspector	
□ L Event Log	
🛱 M Device Manager	
≡ 0 TODO	
🟚 R Resource Manager	
🔄 T Terminal	
🛎 V Build Variants	
~/OneDrive/Documents/Books/Jetp	ack_Compose/JetpackComposeEssentials/ComposeDemo/app

Figure 7-9

Once displayed, the switcher will remain visible for as long as the Ctrl key remains depressed. Repeatedly tapping the Tab key while holding down the Ctrl key will cycle through the various selection options while releasing the Ctrl key causes the currently highlighted item to be selected and displayed within the main window.

In addition to the switcher, navigation to recently opened files is provided by the Recent Files panel (Figure 7-10). This can be accessed using the Ctrl-E keyboard shortcut (Cmd-E on macOS). Once displayed, either the mouse pointer can be used to select an option or the keyboard arrow keys used to scroll through the file name and tool window options. Pressing the Enter key will select the currently highlighted item.



~/OneDrive/Documents/Books/Jetpack_Compose/JetpackComposeEssentials/ComposeDemo/app

Figure 7-10

A Tour of the Android Studio User Interface

7.6 Changing the Android Studio theme

The overall theme of the Android Studio environment may be changed either from the welcome screen using the *Configure -> Settings* option or via the *File -> Settings*... menu option (*Android Studio -> Preferences*... on macOS) of the main window.

Once the settings dialog is displayed, select the *Appearance & Behavior* option followed by *Appearance* in the left-hand panel and then change the setting of the *Theme* menu before clicking on the *Apply* button. The themes available will depend on the platform but usually include options such as Light, IntelliJ, Windows, High Contrast, and Darcula. Figure 7-11 shows an example of the main window with the Darcula theme selected:





To synchronize the Android Studio theme with the operating system light and dark mode setting, enable the *Sync with OS* option and use the drop-down menu to control which theme to use for each mode:

Appearance & Behavior > Appearance	
Theme: IntelliJ Light 🔻 🗹 Sync with OS	¢.
	Preferred Theme
Use custom font: .AppleSystemUIFont	Light
	✓ IntelliJ Light
Accessibility Support screen readers Requires restart	Dark ✓ Darcula High contrast
Use contrast scrollbars	



7.7 Summary

The primary elements of the Android Studio environment consist of the welcome screen and main window. Each open project is assigned its own main window which, in turn, consists of a menu bar, toolbar, editing and design area, status bar, and a collection of tool windows. Tool windows appear on the sides and bottom edges of the main window and can be accessed either using the quick access menu located in the status bar or via the optional tool window bars.

There are very few actions within Android Studio that cannot be triggered via a keyboard shortcut. A keymap of default keyboard shortcuts can be accessed at any time from within the Android Studio main window.

8. Testing Android Studio Apps on a Physical Android Device

While much can be achieved by testing applications using an Android Virtual Device (AVD), there is no substitute for performing real-world application testing on a physical Android device and there are some Android features that are only available on physical Android devices.

Communication with both AVD instances and connected Android devices is handled by the *Android Debug Bridge* (*ADB*). In this chapter, we explain how to configure the adb environment to enable application testing on an Android device with macOS, Windows, and Linux-based systems.

8.1 An overview of the Android Debug Bridge (ADB)

The primary purpose of the ADB is to facilitate interaction between a development system, in this case, Android Studio, and both AVD emulators and Android devices to run and debug applications. ADB allows you to connect to devices either over a WiFi network or directly using a USB cable.

The ADB consists of a client, a server process running in the background on the development system, and a daemon background process running in either AVDs or real Android devices such as phones and tablets.

The ADB client can take a variety of forms. For example, a client is provided in the form of a command-line tool named *adb* located in the Android SDK *platform-tools* sub-directory. Similarly, Android Studio also has a built-in client.

A variety of tasks may be performed using the *adb* command-line tool. For example, a listing of currently active virtual or physical devices may be obtained using the *devices* command-line argument. The following command output indicates the presence of an AVD on the system but no physical devices:

\$ adb devices List of devices attached emulator-5554 device

8.2 Enabling USB debugging ADB on Android devices

Before ADB can connect to an Android device, that device must first be configured to allow the connection. On phone and tablet devices running Android 6.0 or later, the steps to achieve this are as follows:

- 1. Open the Settings app on the device and select the *About tablet* or *About phone* option (on some versions of Android this can be found on the *System* page of the Settings app).
- 2. On the *About* screen, scroll down to the *Build number* field (Figure 8-1) and tap on it seven times until a message appears indicating that developer mode has been enabled. If the Build number is not listed on the About screen it may be available via the *Software information* option. Alternatively, unfold the Advanced section of the list if available.



Figure 8-1

3. Return to the main Settings screen and note the appearance of a new option titled Developer options (on newer versions of Android this option is listed on the System settings screen). Select this option and on the resulting screen, locate the USB debugging option as illustrated in Figure 8-2:





4. Enable the USB debugging option and tap the Allow button when confirmation is requested.

At this point, the device is now configured to accept debugging connections from adb on the development system over a USB connection. All that remains is to configure the development system to detect the device when it is attached. While this is a relatively straightforward process, the steps involved differ depending on whether the development system is running Windows, macOS, or Linux. Note that the following steps assume that the Android SDK *platform-tools* directory is included in the operating system PATH environment variable as described in the chapter entitled *"Setting up an Android Studio Development Environment"*.

8.2.1 macOS ADB configuration

To configure the ADB environment on a macOS system, connect the device to the computer system using a USB cable, open a terminal window and execute the following command to restart the adb server:

```
$ adb kill-server
$ adb start-server
* daemon not running. starting it now on port 5037 *
* daemon started successfully *
```

Once the server is successfully running, execute the following command to verify that the device has been detected:

```
$ adb devices
List of devices attached
74CE00060000001 offline
```

If the device is listed as *offline*, go to the Android device and check for the presence of the dialog shown in Figure 8-3 seeking permission to *Allow USB debugging*. Enable the checkbox next to the option that reads *Always allow from this computer*, before clicking on *OK*.



Figure 8-3

Repeating the *adb devices* command should now list the device as being available:

List of devices attached 015d41d4454bf80c device

If the device is not listed, try logging out and then back into the macOS desktop and, if the problem persists, rebooting the system.

8.2.2 Windows ADB configuration

The first step in configuring a Windows-based development system to connect to an Android device using ADB is to install the appropriate USB drivers on the system. The USB drivers to install will depend on the model of the Android Device. If you have a Google device such as a Pixel phone, then it will be necessary to install and configure the Google USB Driver package on your Windows system. Detailed steps to achieve this are outlined on the following web page:

https://developer.android.com/sdk/win-usb.html

For Android devices not supported by the Google USB driver, it will be necessary to download the drivers provided by the device manufacturer. A listing of drivers together with download and installation information can be obtained online at:

https://developer.android.com/tools/extras/oem-usb.html

With the drivers installed and the device now being recognized as the correct device type, open a Command Prompt window and execute the following command:

adb devices

This command should output information about the connected device similar to the following:

List of devices attached HT4CTJT01906 offline

If the device is listed as *offline* or *unauthorized*, go to the device display and check for the dialog shown in Figure 8-3 seeking permission to *Allow USB debugging*. Enable the checkbox next to the option that reads *Always allow from this computer*, before clicking on *OK*. Repeating the *adb devices* command should now list the device as being ready:

```
List of devices attached HT4CTJT01906 device
```

If the device is not listed, execute the following commands to restart the ADB server:

```
adb kill-server
adb start-server
```

Testing Android Studio Apps on a Physical Android Device

If the device is still not listed, try executing the following command:

android update adb

Note that it may also be necessary to reboot the system.

8.2.3 Linux adb configuration

For this chapter, we will once again use Ubuntu Linux as a reference example in terms of configuring adb on Linux to connect to a physical Android device for application testing.

Physical device testing on Ubuntu Linux requires the installation of a package named *android-tools-adb* which, in turn, requires that the Android Studio user be a member of the *plugdev* group. This is the default for user accounts on most Ubuntu versions and can be verified by running the *id* command. If the plugdev group is not listed, run the following command to add your account to the group:

sudo usermod -aG plugdev \$LOGNAME

After the group membership requirement has been met, the *android-tools-adb* package can be installed by executing the following command:

sudo apt-get install android-tools-adb

Once the above changes have been made, reboot the Ubuntu system. Once the system has restarted, open a Terminal window, start the adb server and check the list of attached devices:

```
$ adb start-server
* daemon not running. starting it now on port 5037 *
* daemon started successfully *
$ adb devices
List of devices attached
015d41d4454bf80c offline
```

If the device is listed as *offline* or *unauthorized*, go to the Android device and check for the dialog shown in Figure 8-3 seeking permission to *Allow USB debugging*.

8.3 Resolving USB connection issues

If you are unable to successfully connect to the device using the above steps, display the run target menu (Figure 8-4) and select the *Troubleshoot Device Connections* option:





The connection assistant will scan for devices and report problems and possible solutions.

8.4 Enabling wireless debugging on Android devices

Follow steps 1 through 3 from section 8.2 above, this time enabling the Wireless Debugging option as shown in Figure 8-5:





Figure 8-6

If the device you are using has a camera, select *Pair device with QR code*, otherwise select the *Pair device with pairing code* option. Depending on your selection, the Settings app will either start a camera session or display a pairing code as shown in Figure 8-7:





With an option selected, return to Android Studio and select the *Pair Devices Using WiFi* option from the run target menu as illustrated in Figure 8-8:

Testing Android Studio Apps on a Physical Android Device





In the pairing dialog, select either *Pair using QR code* or *Pair using pairing code* depending on your previous selection in the Settings app on the device:





Either scan the QR code using the Android device or enter the pairing code displayed on the device screen into the Android Studio dialog (Figure 8-10) to complete the pairing process:

Enter pairing code	
	Enter the 6 digit code shown on the device at 10.0.2.15:43967 to pair.
	0 0 0 0 0 0
	Cancel Pair



If the pairing process fails, try rebooting both the development system and Android device and try again.

8.5 Testing the adb connection

Assuming that the adb configuration has been successful on your chosen development platform, the next step is to try running the test application created in the chapter entitled *"An Example Compose Project"* on the device. Launch Android Studio, open the ComposeDemo project, and verify that the device appears in the device selection menu as highlighted in Figure 8-11:



Figure 8-11

Select the device from the list and click on the run button (the green arrow button located immediately to the right of the device menu) to install and run the app.

8.6 Summary

While the Android Virtual Device emulator provides an excellent testing environment, it is important to keep in mind that there is no real substitute for making sure an application functions correctly on a physical Android device. This, after all, is where the application will be used in the real world.

By default, however, the Android Studio environment is not configured to detect Android devices as a target testing device. It is necessary, therefore, to perform some steps to be able to load applications directly onto an Android device from within the Android Studio development environment either via a USB cable or over a WiFi network. The exact steps to achieve this goal differ depending on the development platform being used. In this chapter, we have covered those steps for Linux, macOS, and Windows-based platforms.