

# Kotlin

## Einstieg und Praxis

# Inhaltsverzeichnis

|          |                                              |    |
|----------|----------------------------------------------|----|
|          | <b>Einleitung</b> .....                      | 13 |
| <b>1</b> | <b>Eigenschaften von Kotlin</b> .....        | 21 |
| 1.1      | Eine neue Sprache .....                      | 21 |
| 1.2      | Kotlin und Android .....                     | 23 |
| 1.3      | Vorteile von Kotlin .....                    | 24 |
| 1.4      | Ziele der Entwicklung von Kotlin .....       | 25 |
| 1.4.1    | Übersichtlichkeit, präziser Ausdruck .....   | 25 |
| 1.4.2    | Sicherheit .....                             | 28 |
| 1.4.3    | Interoperabilität .....                      | 29 |
| <b>2</b> | <b>Arbeiten mit Kotlin</b> .....             | 33 |
| 2.1      | Entwicklungsumgebung einrichten .....        | 33 |
| 2.1.1    | IntelliJ .....                               | 33 |
| 2.1.2    | Android Studio .....                         | 37 |
| 2.2      | Tools .....                                  | 39 |
| 2.2.1    | Kotlin-Compiler .....                        | 40 |
| 2.2.2    | Kotlin-Compiler mit Gradle .....             | 40 |
| 2.2.3    | Integrierte Entwicklungsumgebungen/IDE ..... | 41 |
| 2.2.4    | Konvertierung von existierendem Code .....   | 41 |
| <b>3</b> | <b>Kotlin by example</b> .....               | 43 |
| 3.1      | Hello world! .....                           | 44 |
| 3.2      | Hello null! .....                            | 46 |
| 3.2.1    | Ein erster Test .....                        | 47 |
| 3.2.2    | Eine erste Exception .....                   | 48 |
| 3.2.3    | Eine erste Variable – Null-Sicherheit .....  | 49 |
| 3.2.4    | Let und Lambdas .....                        | 50 |
| 3.2.5    | Eine erste Schleife .....                    | 53 |
| 3.3      | Hello Android! .....                         | 55 |
| 3.4      | Hello Kotlin! .....                          | 59 |
| <b>4</b> | <b>Grundlagen und Daten</b> .....            | 63 |
| 4.1      | Expressions vs. Statements .....             | 63 |

|       |                                                                             |     |
|-------|-----------------------------------------------------------------------------|-----|
| 4.2   | Strukturierung von Code durch Dateien, Klassen, Funktionen und Blöcke ..... | 65  |
| 4.3   | Funktionen .....                                                            | 69  |
| 4.3.1 | Direkte Funktionsaufrufe .....                                              | 70  |
| 4.3.2 | Benutzung von Lambdas .....                                                 | 71  |
| 4.4   | Variablen .....                                                             | 72  |
| 4.4.1 | Deklaration: Immutable vs. Mutable, Zuweisung .....                         | 74  |
| 4.4.2 | Typen und Typsicherheit .....                                               | 77  |
| 4.4.3 | Null-Sicherheit .....                                                       | 77  |
| 4.4.4 | Typinferenz .....                                                           | 84  |
| 4.4.5 | Generics benutzen .....                                                     | 85  |
| 4.4.6 | Boxing .....                                                                | 85  |
| 4.5   | Besonderheiten der Datentypen .....                                         | 88  |
| 4.5.1 | Zahlen .....                                                                | 88  |
| 4.5.2 | Boolean .....                                                               | 90  |
| 4.5.3 | Chars und Strings – Zeichen und Zeichenketten .....                         | 90  |
| 4.5.4 | Objekte und Any .....                                                       | 93  |
| 4.5.5 | Arrays und Listen .....                                                     | 94  |
| 4.5.6 | Maps .....                                                                  | 98  |
| 4.5.7 | Ranges .....                                                                | 99  |
| 5     | <b>Grundstrukturen</b> .....                                                | 101 |
| 5.1   | Kontrollstrukturen .....                                                    | 101 |
| 5.2   | Verzweigungen mit if .....                                                  | 101 |
| 5.3   | Verzweigung mit when .....                                                  | 103 |
| 5.4   | Schleifen mit for und while .....                                           | 105 |
| 5.4.1 | for-Schleifen .....                                                         | 106 |
| 5.4.2 | while-Schleifen .....                                                       | 109 |
| 5.4.3 | repeat .....                                                                | 110 |
| 5.4.4 | break/named break und continue .....                                        | 110 |
| 5.4.5 | Funktionsbasierte Alternativen .....                                        | 112 |
| 5.5   | Exception Handling mit try/catch .....                                      | 114 |
| 5.6   | Annotations .....                                                           | 116 |
| 5.7   | Datentypen .....                                                            | 118 |
| 5.7.1 | Type-Casting .....                                                          | 119 |
| 5.7.2 | Type-Casting und null .....                                                 | 120 |
| 5.7.3 | Smart Casts .....                                                           | 121 |
| 5.7.4 | Type Aliases .....                                                          | 122 |

|          |                                                |            |
|----------|------------------------------------------------|------------|
| <b>6</b> | <b>Funktionen und Lambdas</b>                  | <b>125</b> |
| 6.1      | Funktionale Entwicklung                        | 125        |
| 6.2      | Syntax von Funktionen                          | 127        |
| 6.2.1    | Kurzformen von Funktionsdefinitionen           | 129        |
| 6.2.2    | Parameter                                      | 129        |
| 6.2.3    | Variable Anzahl von Parametern                 | 131        |
| 6.2.4    | Rückgabe                                       | 132        |
| 6.2.5    | Destrukturierende Deklarationen                | 133        |
| 6.2.6    | Kombination mehrerer Funktionsaufrufe          | 135        |
| 6.3      | Höhere Funktion und Lambdas                    | 136        |
| 6.3.1    | Funktionen als Parameter                       | 136        |
| 6.3.2    | Funktionstyp für Variablen                     | 137        |
| 6.4      | Chaining von Lambdas                           | 138        |
| 6.5      | Rekursive Funktionen                           | 138        |
| 6.6      | Spezielle Funktionen                           | 139        |
| 6.6.1    | Lokale Funktionen                              | 139        |
| 6.6.2    | Inline-Funktionen                              | 141        |
| 6.6.3    | Infix-Funktionen                               | 143        |
| 6.6.4    | Actual-Funktionen                              | 143        |
| 6.6.5    | Überladene Operatoren                          | 144        |
| <b>7</b> | <b>Packages, Klassen und Objekte</b>           | <b>145</b> |
| 7.1      | Architektur und Entwurf                        | 145        |
| 7.2      | Packages, Klassen und Interfaces               | 145        |
| 7.2.1    | Klassen und Interfaces organisieren das Modell | 146        |
| 7.2.2    | Klassen und Interfaces, Vererbung              | 146        |
| 7.2.3    | Packages organisieren das Projekt              | 149        |
| 7.2.4    | Sichtbarkeitsmodifizier                        | 151        |
| 7.2.5    | Import                                         | 152        |
| 7.3      | Von der Klasse zum Objekt – Meet the members   | 154        |
| 7.3.1    | Konstruktoren und Initializer                  | 154        |
| 7.3.2    | Properties                                     | 158        |
| 7.3.3    | Custom Getter/Setter                           | 159        |
| 7.3.4    | Methoden                                       | 161        |
| 7.3.5    | Innere Klassen                                 | 163        |
| 7.3.6    | Companion Objects                              | 165        |
| 7.4      | Objekt-Anweisungen                             | 166        |
| 7.4.1    | Singletons via Objekt-Deklaration              | 166        |

|       |                                                                      |     |
|-------|----------------------------------------------------------------------|-----|
| 7.4.2 | Anonyme Klassen via Objekt-Ausdruck .....                            | 168 |
| 7.4.3 | Object-Statements bei Rückgaben .....                                | 169 |
| 7.5   | Datenklassen .....                                                   | 170 |
| 7.6   | Generics .....                                                       | 171 |
| 7.6.1 | Generische Klassen .....                                             | 171 |
| 7.6.2 | Beschränkung der Typen: Invariant, Kovariant,<br>Kontravariant. .... | 173 |
| 7.6.3 | Generische Methoden .....                                            | 177 |
| 7.6.4 | Reified parameters .....                                             | 178 |
| 7.7   | Enum classes .....                                                   | 179 |
| 7.8   | Sealed classes .....                                                 | 181 |
| 7.9   | Extensions. ....                                                     | 182 |
| 7.10  | Destrukturierende Deklarationen .....                                | 184 |
| 8     | <b>Testing</b> .....                                                 | 187 |
| 8.1   | Einführung. ....                                                     | 187 |
| 8.1.1 | Unit-Tests .....                                                     | 188 |
| 8.1.2 | Integrationstests .....                                              | 188 |
| 8.1.3 | Testsuite .....                                                      | 189 |
| 8.1.4 | Mocking und Dependency Injection .....                               | 189 |
| 8.1.5 | Konkret: Wie werden Tests geschrieben? .....                         | 190 |
| 8.2   | Erster JUnit-Test .....                                              | 191 |
| 8.3   | Einrichten eines Testprojekts .....                                  | 193 |
| 8.4   | Praxisbeispiel: Verkleinern von Funktionen .....                     | 194 |
| 8.5   | Das Testresultat lesen .....                                         | 200 |
| 8.6   | Der Test von Hello world! .....                                      | 202 |
| 8.7   | Test einer Klasse .....                                              | 204 |
| 8.8   | Parametrisierte Tests .....                                          | 207 |
| 8.9   | Testing Kotlin-Style .....                                           | 208 |
| 8.9.1 | Testkomponenten .....                                                | 208 |
| 8.9.2 | Spek .....                                                           | 210 |
| 8.9.3 | Kotlintest.io .....                                                  | 210 |
| 8.9.4 | kotlin.test: Unit-Tests in Multiplattform-Projekten .....            | 211 |
| 9     | <b>Kotlin für Fortgeschrittene</b> .....                             | 215 |
| 9.1   | Alles zusammenbringen .....                                          | 215 |
| 9.2   | Ziellplattformen .....                                               | 216 |
| 9.3   | Kotlin-Standard .....                                                | 217 |
| 9.3.1 | run, let, apply, with, also .....                                    | 218 |

|           |                                                    |            |
|-----------|----------------------------------------------------|------------|
| 9.3.2     | Takeif/Takeunless .....                            | 223        |
| 9.3.3     | Collections, Arrays und Listen. ....               | 224        |
| 9.3.4     | Operationen auf allen iterierbaren Elementen. .... | 226        |
| 9.3.5     | Texttools .....                                    | 233        |
| 9.3.6     | Reflection .....                                   | 234        |
| 9.4       | Delegation .....                                   | 236        |
| 9.4.1     | Property Delegation .....                          | 237        |
| 9.4.2     | Class Delegation .....                             | 241        |
| 9.5       | Service Location/Dependency Injection .....        | 246        |
| 9.5.1     | Warum eigentlich? .....                            | 247        |
| 9.5.2     | Service Locator .....                              | 248        |
| 9.5.3     | Dependency Injection .....                         | 251        |
| 9.6       | Coroutines und Nebenläufigkeit .....               | 253        |
| 9.6.1     | Launch und Scopes .....                            | 253        |
| 9.6.2     | Async, Await und Deferred .....                    | 257        |
| 9.6.3     | Producer, Consumer und Actor .....                 | 258        |
| 9.7       | Contracts .....                                    | 258        |
| 9.8       | Idiomatischer Code .....                           | 261        |
| 9.8.1     | Unveränderliche/immutable Variablen .....          | 261        |
| 9.8.2     | Kurzformen für null. ....                          | 262        |
| 9.8.3     | Kurzschreibweisen für Funktionen .....             | 262        |
| 9.8.4     | Nutzung von Class Properties .....                 | 262        |
| 9.8.5     | Chaining von nicht chainbaren Aufrufen. ....       | 262        |
| 9.8.6     | Extensions nutzen .....                            | 263        |
| 9.8.7     | DSL mit Lambdas .....                              | 263        |
| 9.8.8     | Infix-Notation für klare Sätze .....               | 263        |
| <b>10</b> | <b>Kotlin für Android</b> .....                    | <b>265</b> |
| 10.1      | Android <3 Kotlin .....                            | 265        |
| 10.2      | Android-Komponenten .....                          | 266        |
| 10.2.1    | Package-Level und Überblick. ....                  | 266        |
| 10.2.2    | Klassenlevel. ....                                 | 267        |
| 10.3      | Bestandteile des Android-Frameworks .....          | 270        |
| 10.3.1    | Activities und Fragmente. ....                     | 271        |
| 10.3.2    | Eine minimale App .....                            | 272        |
| 10.3.3    | Views, Rendering und Layouting .....               | 275        |
| 10.3.4    | Der Lifecycle .....                                | 282        |
| 10.3.5    | Android Architecture Components .....              | 283        |

|         |                                                      |     |
|---------|------------------------------------------------------|-----|
| 10.4    | Architektur für größere App-Projekte .....           | 285 |
| 10.5    | Das Architekturmodell MVP(I) .....                   | 286 |
| 10.5.1  | Der Vertrag .....                                    | 287 |
| 10.5.2  | Initialisierung .....                                | 288 |
| 10.5.3  | Daten laden und initiale UI erstellen .....          | 289 |
| 10.5.4  | Mehrere Presenter/Komponenten .....                  | 292 |
| 10.5.5  | Updates der UI und Reagieren auf Dateneingaben ..... | 293 |
| 10.6    | Das Architekturmodell MVVM .....                     | 296 |
| 10.6.1  | MVVM-Komponenten anlegen .....                       | 297 |
| 10.6.2  | Initialisierung .....                                | 297 |
| 10.6.3  | Daten laden und initiale UI erstellen .....          | 298 |
| 10.6.4  | Updates der UI und Reagieren auf Dateneingaben ..... | 301 |
| 10.7    | MVP oder MVVM? It depends ... ..                     | 302 |
| 10.8    | Erweiterung von Android-Klassen .....                | 302 |
| 10.9    | Android und Coroutines .....                         | 303 |
| 10.10   | Delegation für Android .....                         | 304 |
| 10.10.1 | Property Delegation .....                            | 304 |
| 10.10.2 | Klassendelegation .....                              | 306 |
| 11      | <b>Zielpattformen für Kotlin</b> .....               | 311 |
| 11.1    | Multiplattform-Logik .....                           | 311 |
| 11.2    | Java .....                                           | 312 |
| 11.2.1  | Java-Libraries .....                                 | 312 |
| 11.2.2  | Reflection .....                                     | 312 |
| 11.2.3  | Streams (Java 8) .....                               | 314 |
| 11.2.4  | Java Collections .....                               | 314 |
| 11.2.5  | Java Annotations .....                               | 314 |
| 11.2.6  | Annotation Use-Site Target .....                     | 317 |
| 11.3    | JavaScript in Kotlin .....                           | 318 |
| 11.3.1  | Einrichtung oder: Wie geht denn das? .....           | 318 |
| 11.3.2  | Grundsätzliches zu Kotlin und JS .....               | 318 |
| 11.3.3  | Standalone JS-App .....                              | 320 |
| 11.4    | JS-Standard-Bibliothek .....                         | 320 |
| 11.4.1  | Dynamic .....                                        | 321 |
| 11.4.2  | Promises .....                                       | 322 |
| 11.4.3  | Browser .....                                        | 322 |
| 11.4.4  | JSON-Parsing .....                                   | 324 |
| 11.4.5  | Nutzung von JS-Bibliotheken/-Aufrufen .....          | 326 |

|        |                                                               |     |
|--------|---------------------------------------------------------------|-----|
| 11.5   | Konvertierung von existierendem Java-Code zu JS. ....         | 327 |
| 11.6   | Konvertierung und Bearbeitung von existierendem JS-Code. .... | 328 |
| 11.7   | Multiplattform-Projekte .....                                 | 328 |
| 11.7.1 | Ein Multiplattform-Projekt beginnen .....                     | 330 |
| 11.7.2 | Expected/Actual .....                                         | 333 |
| 11.8   | Native .....                                                  | 338 |
| 12     | <b>Rück- und Ausblick</b> .....                               | 339 |
| A      | <b>Referenztabellen</b> .....                                 | 341 |
| A.1    | Factories .....                                               | 341 |
| A.1.1  | Factories für Arrays .....                                    | 341 |
| A.1.2  | Factories für Listen und Maps .....                           | 342 |
| A.2    | Konvertierung von Listen/Arrays .....                         | 342 |
| A.3    | Operationen auf allen iterierbaren Elementen .....            | 343 |
| A.3.1  | Zugriff auf einzelne Elemente .....                           | 343 |
| A.3.2  | Zugriff auf Maps .....                                        | 344 |
| A.3.3  | Iteration über alle oder viele Elemente .....                 | 344 |
| A.3.4  | Manipulation von Listen .....                                 | 345 |
| A.3.5  | Manipulation von Maps .....                                   | 346 |
|        | <b>Stichwortverzeichnis</b> .....                             | 347 |



# Einleitung

Fast jeder kennt das unangenehme Gefühl, einer wildfremden Person zuzuwin-  
ken, weil man kurz dachte, die freundliche Geste gälte einem selbst. Galt sie natür-  
lich nicht, sondern der Person, die sich in direkter Linie hinter der eigenen  
Schulter im toten Winkel versteckte. Auf der Google I/O 2017 habe ich erlebt, wie  
sich ein ganz ähnliches Empfinden einstellt, wenn man eine halbe Stunde lang in  
eine Fernsehkamera blickt, die in etwas mehr als Armeslänge vor einem positio-  
niert ist. Selbstverständlich wusste ich, dass es keinen Grund gab, mich in dieser  
Weise zu portraituren. Doch nach fünf Minuten ... und weiteren zehn ... es konnte  
doch nicht?

Konnte es natürlich nicht. Es ging um einen 17-jährigen High-School Studenten,  
den der Lebensweg seiner Eltern als Einwanderer aus Afghanistan und die in  
Amerika erlebte Hilfsbereitschaft inspiriert hat, selbst etwas zurückzugeben. In  
der Reihe vor mir. Sein Projekt verbesserte ohne teure Hardware die Aussagefähig-  
keit von Mammographien mithilfe von künstlicher Intelligenz. Die von ihm pro-  
grammierten Routinen erkennen Muster in den Bildern und können so auch ohne  
medizinisches Fachpersonal präzisere Vorabdiagnosen zur Verfügung stellen.<sup>1</sup>

Eine knappe halbe Stunde zuvor gab es eine Ankündigung, die mir diesen Tag  
auch ohne die bemerkenswert große Kamera im Gedächtnis gehalten hätte: Die  
Ankündigung von Stephanie Saad Cuthbertson (Director Product Management  
Android zu der Zeit), Kotlin zu einer vollständig unterstützten Programmierspra-  
che für Android zu erklären und mit dem Entwickler *JetBrains* eine Gesellschaft  
zur Weiterentwicklung der Sprache zu gründen. Kotlin ist eine vergleichsweise  
junge Programmiersprache, deren erste stabile Version 2016 erschien. Seit 2011  
wird sie von einem Team des Herstellers der Entwicklungsumgebung *IntelliJ* wei-  
terentwickelt und gepflegt. Hier besteht auch die offensichtliche Verbindung zu  
Java und Android, denn auch die von Google primär unterstützte *IDE* (Integrated  
Development Environment oder Entwicklungsumgebung) *Android Studio* basiert  
auf *IntelliJ*.

Dadurch war auch gewährleistet, dass es in Bezug auf die Tools keiner großen Um-  
gewöhnung bedurfte. Und so haben sicherlich einige Android-Entwickler nach

---

1 Im Video der Keynote bei 1:50:50 können sich die Leserinnen und Leser dieses Buchs ein Bild  
von mir im Jahr 2017 machen. Ganz Kamera-Unprofi erkennt man auch den Moment, in dem  
ich begriffen habe, dass es ganz sicher nicht um mich geht.

dieser Ankündigung 2017 ihre ersten Schritte in Kotlin mithilfe der automatischen Konvertierungsfunktion gemacht und schnell eines erkannt: Die neue Sprache löst sofort Probleme im bestehenden Code ihrer Anwendung:

- Der Code wird kürzer und übersichtlicher
- Der Code wird sicherer
- Es ist kein Problem, Kotlin und Java miteinander in einem Projekt einzusetzen
- Keine Semikolons(!)

Oder, um es mit den Worten von Google und JetBrains zu sagen: Wichtige Merkmale von Kotlin sind: Es ist effektiv durch präzise Schreibweisen (*concise*), sicher (*safe*) und gut mit verschiedenen Umgebungen zu integrieren (*interoperable*).

Wie das in der Praxis aussieht und warum ich schon nach den ersten Arbeitstagen mit Kotlin kein Java mehr sehen mochte, versuche ich in diesem Buch zu erklären.

## Ausrichtung des Buchs

Dieses Buch soll Entwicklern einen Einstieg in die Programmiersprache Kotlin ermöglichen. Es konzentriert sich auf die Grundlagen der Sprache und ihre Strukturen, Befehle und Sprachfeatures. Zusätzlich zur Erläuterung der ursprünglichen Aufgabe als ein verbesserter Java-Ersatz wird das aktuelle Haupteinsatzfeld Android behandelt und wie Kotlin die Android-Entwicklung verbessern – oder neutral: verändern – kann. Dabei werden aber Kenntnisse von Android vorausgesetzt, um den Fokus auf die Sprache Kotlin beizubehalten.

Das vorliegende Buch möchte dazu motivieren, vor einem Volleinstieg in Kotlin die Konvertierung Schritt für Schritt zu probieren. Das Stichwort lautet Interoperabilität. Auch in einem großen Java-Spring-Projekt können einzelne Features oder Klassen in Kotlin geschrieben oder in der Webanwendung einzelne Controller oder Datenklassen ergänzt werden. Bei den Beispielen benutze ich wenig Framework-Code und fokussiere mich auf die Umsetzung der Logik in Kotlin. Wenn, dann wird vor allem Android verwendet.

Generell ist es aber gut möglich, auch sein favorisiertes JavaScript- oder TypeScript-Framework weiterzunutzen. Je elaborierter die Build- und Deployment-Prozesse jedoch sind, desto weniger kann ich als App-Entwickler Details der Umsetzung beleuchten.

Darüber hinaus wagt das Kapitel 11 einen Ausflug in die Welt der Cross-Plattform-Entwicklung. Das ist wichtig für alle, deren Code in Zukunft aus einer Kotlin-Codebasis nicht nur auf Android-Smartphones **oder** im Browser laufen soll.

## Zielgruppe

Dieses Buch ist für alle geschrieben, die sich für die Entwicklung mit Kotlin interessieren und die bereits grundsätzliche Kenntnisse in der Softwareentwicklung haben.

Dazu gehört zum Beispiel, dass Sie grundlegende Quelltext-Elemente wie Variablen, Konstanten, Kontrollstrukturen und Funktionen kennen und Ihnen Begriffe aus der objektorientierten Entwicklung wie Klasse, Objekt und Instanz geläufig sind. Programmierparadigmen und Muster der Softwareentwicklung werden ebenfalls nicht ausführlich erklärt, sondern vor allem dort erwähnt, wo es zum Verständnis der Unterschiede zwischen Kotlin und seinen Alternativen dient. Einblicke in die Programmiersprache Java helfen, sind aber nicht zwingend erforderlich.

Die meisten Entwickler sammeln ihre ersten Programmiererfahrungen in Kotlin nicht als Einsteiger in die Softwareentwicklung. Sie gehen mit Erwartungen aus bekannten Sprachen oder sogar mit existierendem Code an die Arbeit.

Mit Kotlin lässt sich in dieser Situation besonders gut arbeiten, denn es erfordert für viele Entwickler (Java, Android) keinen Umstieg auf eine andere Entwicklungsumgebung und verträgt sich mit bestehenden Infrastrukturen. Durch die hohe Flexibilität im Bereich der Interoperabilität ist es eine gute Strategie, in bestehendem Code einfach einmal einen konkreten Anwendungsfall zu ergänzen oder einzelne Klassen zu ersetzen. Damit können Entwicklungsteams eine informierte Entscheidung treffen, ohne zwangsläufig vor einem kompletten Neuanfang zu stehen.

## Aufbau des Buchs

Der Aufbau dieses Buchs folgt meinen Erfahrungen als Hands-On-Entwickler und Geschäftsführer einer Agentur, in der Softwareentwickler verschiedener Erfahrungsstufen arbeiten.

In unserem Team haben alle Kotlin auf der Basis von Android kennengelernt. Sie waren und sind erfahrene Java-Entwickler und Experten in der Umsetzung von Apps. Wichtige Muster aus der App-Entwicklung und auch die Best Practices in Android waren also von vornherein bekannt. Folgerichtig handelte es sich bei den ersten Experimenten denn auch eher um Java ohne Semikolons. Auf Basis einer fundierten Kenntnis der APIs und Tools für die Android-Entwicklung hat sich erst im Verlauf der Projekte ein tieferes Verständnis für die weitergehenden Möglichkeiten von Kotlin entwickelt. Das scheint mir typisch für eine Sprache, die entwickelt wurde, um den Unzulänglichkeiten einer existierenden Vorgängerin zu begegnen. Ich werde deshalb auf den folgenden Seiten ähnlich vorgehen.

In **Kapitel 1** gebe ich zunächst einen Überblick über die Eigenschaften und Vorteile von Kotlin, seine Konzepten und die positiven Effekte auf die eigene Codebasis.

In **Kapitel 2** erläutere ich kurz die zur Verfügung stehenden Tools, die ich für die ersten Schritte empfehle.

**Kapitel 3** enthält kommentierte Beispiele, um zunächst ein Gefühl für die Sprache zu geben und ihre Möglichkeiten zu erkunden. Für diejenigen Leser, die bereits Vorkenntnisse in Android oder zum Beispiel Java haben, wird dieser Einstieg vielleicht für die ersten eigenen Versuche ausreichen. Einsteiger können dieses Kapitel auch erst mal überspringen. In den darauffolgenden Kapiteln folgt die systematische Betrachtung der einzelnen Sprachelemente.

**Kapitel 4** stellt die grundlegende Syntax von Kotlin in den Mittelpunkt, und konzentriert sich auf die Grundstrukturen von Programmen und die Datentypen.

### **Tipp: Schnelleinstieg mit Kotlin KOANS**

Wer direkt sehen möchte, wie Kotlin funktioniert, dem sei schon an dieser Stelle <https://kotlinlang.org> empfohlen. Unter dem Menüpunkt LEARN finden sich dort die sogenannten *Kotlin Koans* – kleine Übungen, die direkt im Browser kompiliert und getestet werden. Darüber hinaus zeigen sie auch, wie man von Anfang an mit Tests sinnvolle Validierungen der eigenen Überlegungen durchführen kann. Insofern ein Must-Read (and try) für den angehenden Kotlin-Entwickler.

Auch die Beispiele aus dem Buch können aus dem GitHub-Repository dorthin kopiert, ausprobiert und variiert werden.

In **Kapitel 5** werden die Kontrollstrukturen vorgestellt. Es geht also um Verzweigen und Wiederholen – und weitere daran angelehnte Sprachfeatures.

Das **Kapitel 6** widmet sich den Funktionen und ihren verschiedenen Sonderformen. Ich werde dort eine längere Zeit mit höheren Funktionen und Lambdas verbringen, aber auch die Grundlagen von Funktionen ausführlich erklären.

Darauf folgt in **Kapitel 7** ein ausführlicher Block zu den objektorientierten Anteilen von Kotlin. Hier geht es um alle Besonderheiten von Klassen, Objekten und Funktionen im Zusammenspiel. Dafür wird neben einzelnen Code-Schnipseln ein erstes Projekt vorgestellt.

Dieses Projekt beinhaltet eine Expression-Bibliothek, die in der Lage ist, mathematische Ausdrücke zu interpretieren und auszurechnen. Bislang liegt dieser Open-Source-Code in Objective-C, Swift und Java vor, in diesem Kapitel entsteht eine Kotlin-Version sowie eine Kommandozeilen-Anwendung, die sie nutzbar macht.

**Kapitel 8** ist dem Thema automatische Tests in Kotlin gewidmet. Für die Nutzung der Test-Frameworks werden die Inhalte der vorherigen Kapitel als Handwerkszeug benötigt. Die Tests selbst beziehen sich vor allem auf das Testen von Funktionen und Klassen.

**Kapitel 9** steigt tiefer in die komplexeren Konzepte von Kotlin ein, die auf dieser Basis aufbauen. Thema ist hier die Organisation von größeren Projekten durch Architekturmuster. Die Nebenläufigkeit mit Kotlin-Coroutines nimmt einen guten Teil des Kapitels ein und last but not least bespreche ich die Veränderungen im Denken und im Code, die sich aus der Kombination von funktionalem und objekt-orientiertem Stil ergeben.

**Kapitel 10** ist Android gewidmet – hier entsteht eine vollständige App in Kotlin. Auch wenn das Kapitel mit den Grundlagen beginnt, ist es hier nicht möglich, alle Aspekte und Bibliotheken von Android zu behandeln. Der Schwerpunkt liegt auf moderner Architektur und den Anteilen, die sich durch Kotlin besser umsetzen lassen.

Kotlin ist seit der Version 1.2 in Bezug auf Interoperabilität einen Schritt weitergekommen und bietet auch Optionen zum Kompilieren in JavaScript und nativen iOS-Code. Dazu mehr in **Kapitel 11**.

## Downloads zum Buch

Im Repository <https://github.com/kotlin-karlszwilius> finden sich Android-Beispiele, die Kotlin-Portierung der Expression-Bibliothek und die Listings aus den anderen Kapiteln zum Ausprobieren, mit Sourcecode-Dokumentation.

Wer mit GitHub nicht vertraut ist, kann den Großteil der Inhalte auch über die Website des Verlags herunterladen: [www.mitp.de/853](http://www.mitp.de/853)

## Über die Projekte in diesem Buch

Es gibt im Buch kein durchgehendes Projekt, mit dem alle verschiedenen Bereiche von Kotlin gestreift werden. Vielmehr habe ich aus der Praxis verschiedene Versatzstücke aus Projekten herausgelöst, die einzelne Schritte und Techniken der jeweiligen Kapitel illustrieren. So können Sie sich auch einzelne Themen im Buch ansehen, ohne sich in ein großes Beispielprojekt eindenken zu müssen.

Am Anfang des Buchs stehen noch die einzelnen Anweisungen und Ausdrücke im Mittelpunkt, hier handelt es sich um einfach gewählte Beispiele.

Damit ich in der Lage bin, auch komplexere Aufgaben zu zeigen, benutze ich eine Open-Source-Bibliothek, die ich einmal von der Kommandozeile und später auch in einer App verwende. Sie stellt eine vollwertige Sprache zum Lösen von Ausdrü-

cken innerhalb von Formularen oder anderen Anwendungen zur Verfügung. Ihr Name ist NRExpressionLib, sie stammt vom iOS-Entwickler Nikolai Ruhe aus Berlin – wir haben in mehreren Projekten zusammengearbeitet. Da unsere Firma sie bereits von Swift nach Java portiert hat, gehe ich nun einen Schritt weiter und konvertiere sie in die Sprache Kotlin. Diese Aufgabe ist kein klassischer Einstiegspunkt, doch die Konvertierung, auch mithilfe der automatischen Konvertierungsfunktion, führte auch mich zu einigen interessanten Herausforderungen, an denen sich zum Beispiel die Inkompatibilitäten zwischen der Handhabung von *Generics* und den strengen Regeln für Typen bei Kotlin zeigen lassen.

Ein zweites Projekt stammt aus der Praxis und bildet eine Bibliothek von Meditationsübungen ab. Aufgaben wie Navigation und die Darstellung von Listen kann ich anhand dieser Listings einfacher demonstrieren. Diese App steht nicht als vollständiges Codebeispiel zur Verfügung, sondern lediglich in Auszügen, bzw. einzelnen Komponenten, die ausprobiert werden können. Sie ist aber im Play Store als »Key Meditation« kostenlos verfügbar.

Nicht alle Listings im Buch stammen aus diesen beiden Projekten. Einige stehen einfach für sich oder kommen in leicht abgewandelter Form aus dem Arbeitsalltag eines App-Entwicklers.

- Die Codebeispiele im Buch sind daher Auszüge aus längeren Listings, die auf der Seite des Verlags unter [www.mitp.de/853](http://www.mitp.de/853) und im öffentlichen Repository unter <https://github.com/kotlin-karlszwilius> zur Verfügung stehen.
- Auf eine vollständige Befehlsreferenz habe ich zugunsten der bekannten Online-Quellen verzichtet, der Anhang A enthält aber kürzere Übersichten zu den Standardpackages, die sich auch plattformübergreifend verwenden lassen.
- Eine Mischung von Deutsch und Englisch lässt sich nicht ganz vermeiden. Bei Fachbegriffen gebe ich den englischen Versionen den Vorzug und verwende eingedeutschte oder übersetzte Bezeichner nur dort, wo sie vergleichbar allgemeingültig sind.

## Was nicht in diesem Buch steht

Jeder Autor entscheidet über Schwerpunkte und Inhalte eines Buchs, doch für den einen oder anderen mag es wichtiger sein, welche Themen **nicht** behandelt werden. Für mich sind dies in aller Kürze: keine RX-basierte Entwicklung, so gut wie keine experimentellen Features (mit Ausnahmen bei Coroutines und Contracts), kein *Kotlin/Native* und nur wenig über Multiplattform-Projekte.

Kotlin ist eine Sprache, deren Entwicklung noch in vollem Gang ist. Dabei werden vor allem auch im Bereich der Interoperabilität immer wieder neue Welten erschlossen. Mit der Version 1.2 ist *Kotlin/Native* dazugekommen, ein Setup für die Anbindung verschiedener Umgebungen, die auf C und moderneren Ableitungen

davon basieren. Damit ist theoretisch ein Einstieg in die Entwicklung von iOS- und Desktop-Varianten möglich. Hierfür sind zum jetzigen Zeitpunkt noch sehr viel Wissen und Basteltalent in den ursprünglichen Sprachen notwendig, das mir fehlt. Ich freue mich aber auf weiteren Fortschritt in diesem Bereich.

Das zweite Thema, das lediglich gestreift wird, sind Multiplattform-Projekte. Tatsächlich ist ein Projekt in einem solchen Szenario umgesetzt, doch die Unterstützung ist auch in der Version 1.3 noch als experimentell gekennzeichnet. Aus diesem Grund habe ich mich entschieden, auch diesen Teil nur anzureißen und für eine spätere Ergänzung aufzuheben.

Zuletzt habe ich persönlich keine starke Zuneigung zur reaktiven Programmierung mit *RxJava* oder *RxKotlin*, auch wenn sie ihre Vorteile hat und auch regelmäßig Teil von datenintensiven Projekten ist. In der vorliegenden Ausgabe war mir der Fokus auf den Kotlin-eigenen Mechanismen im Bereich Coroutines, Observer und Mapping wichtiger. Für Android habe ich die Vorstellung von LiveData und ViewModel vorgezogen, die in dem gleichen Bereich im Einsatz sind, der heute häufig von Rx-Komponenten abgedeckt wird.

## Weiterentwicklung von Kotlin

Ein wichtiges Anliegen habe ich in Bezug auf Aktualität und Updates der Sprache: Selbstverständlich werden sich von der Erstellung dieses Manuskripts bis zur Veröffentlichung Dinge ändern.

Insbesondere im Bereich von Bibliotheken verändern sich Möglichkeiten und APIs schnell und häufig. Zwar habe ich darauf geachtet, welche Veränderungen zur Erstellung des Manuskripts bereits absehbar waren, doch bereits mit dem nächsten größeren Update können sich auch grundlegende Mechanismen ändern.

Als Beispiel seien hier die Coroutines genannt, die mit der Version 1.2 eingeführt wurden und ein Ersatz für Java-basiertes Multithreading sind. Daher möchte dieses Buch vor allem die Grundlagen vermitteln, mit denen Sie funktionierenden, gut strukturierten und durchdachten Kotlin-Code erstellen können.

## Über den Autor

Meine eigene Ausbildung nennt sich Medienwirt – ein Diplomstudium, das mir weitreichende Freiheiten und Interpretationsmöglichkeiten gewährt hat. So konnte ich die Programmierung, bis dahin Hobby und Schulfach, auch an der Universität weiterverfolgen, dann unter dem Stichwort »Neue Medien«. Heute arbeite ich als Geschäftsführer der Agentur *Karlmax Berlin* an verschiedenen Kundenprojekten im Bereich Apps.

Meine tägliche Arbeit hat bis vor Kurzem auch noch stark in den Projekten stattgefunden, sodass ich auch mit Kotlin dort meinen ersten Kontaktpunkte hatte.

Unsere Agentur hatte das Glück, dass wir mit zwei Kunden zum Zeitpunkt der Bekanntgabe von Kotlin als offizieller Entwicklungssprache für Android bereits über ein Jahr an Projekten mit Kotlin arbeiten durften. Die positiven Einflüsse auf eine Android-App kannten wir so bereits durch konkrete Projekte. Nicht zuletzt mit dieser Erfahrung im Rücken möchte ich alle Entwickler aus dem Java- und Android-Umfeld einladen, sich mit Kotlin zu beschäftigen.



# Arbeiten mit Kotlin

## 2.1 Entwicklungsumgebung einrichten

Um die ersten Schritte mit den Beispielen der kommenden Kapitel ausführen zu können, ist jetzt der Zeitpunkt, eine entsprechende Umgebung zu konfigurieren, um Kotlin auszuprobieren. Meine Empfehlung ist es, hierzu auf eine der frei verfügbaren Entwicklungsumgebungen zu setzen.

Da die ersten Schritte in einem neuen Tool eventuell nicht ganz selbsterklärend sind, stelle ich die beiden wichtigsten Umgebungen vor und zeige sie bis zum ersten minimalen funktionierenden Beispiel. Nun ändern sich Softwareversionen in der Regel schneller als Bücher, doch die grundlegenden Fragen der Organisation haben höchstwahrscheinlich noch eine Weile Bestand, auch wenn die Screens sich bis zur Drucklegung eventuell in Details schon wieder geändert haben.

Es besteht die Wahl zwischen zwei Varianten:

- *IntelliJ Community Edition*: Für alle, deren Fokus nicht direkt auf Android liegt, oder die eventuell auch komplexere Cross-Plattform-Projekte ausprobieren möchten
- *Android Studio*: Für diejenigen, deren Projekte auf Android beschränkt sind

### 2.1.1 IntelliJ

Nach der Auswahl PROJEKT > NEU stehen verschiedene Optionen für Kotlin-Projekte zur Verfügung.

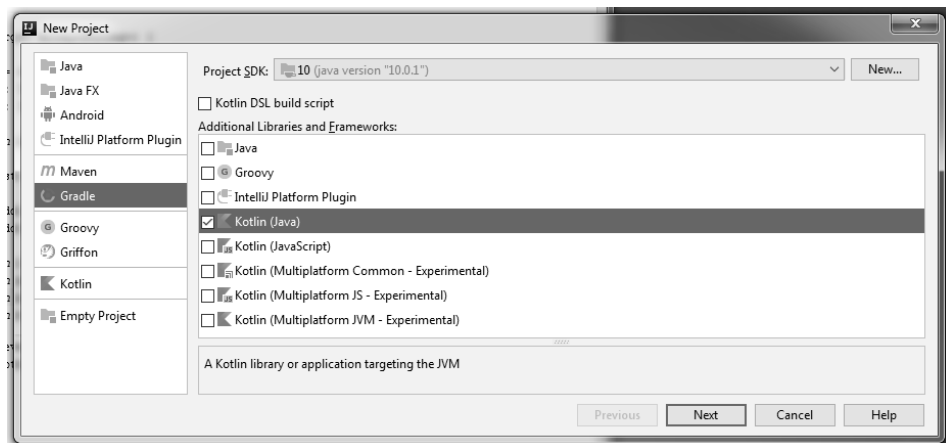
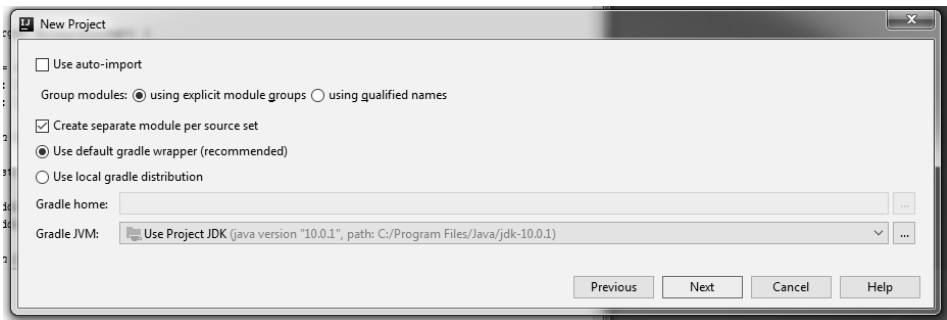


Abb. 2.1: Projektassistent von IntelliJ

Für die Beispielprojekte in den ersten Kapiteln reicht in der Regel das einfache KOTLIN(JAVA)-Projekt aus. In den späteren Kapiteln wird dann auch die Variante für JavaScript, beziehungsweise ein Cross-Plattform-Projekt, verwendet. Wie an den Bezeichnungen bereits ersichtlich ist, handelt es sich zum Teil um experimentelle Features. Diese sind nutzbar, es ist jedoch nicht garantiert, dass nach einem Update der Sprache oder der Tools alles noch läuft wie zuvor.

Im nächsten Schritt muss das Kind einen Namen bekommen. Wer Maven (ein anderes Tool zur Buildorganisation, vergleichbar mit Gradle) nutzt, wird die Einteilung in *Artifact-ID*, *Group-ID* und *Version* kennen. Die Group-ID kann leer bleiben, die Artifact-ID sollte den Projektnamen erhalten, und die Version kann auch zunächst auf dem Default-Wert stehen bleiben.



**Abb. 2.2:** Projektdetails

Die Projekte in IntelliJ werden standardmäßig mit Gradle konfiguriert und gebaut. Daher ist dieser Schritt wichtig. Im Normalfall sollte hier der Gradle-Wrapper aus der Basisinstallation gute Dienste tun.

Es kann im Einzelfall vorkommen, dass sich der aktuelle Wrapper nicht mit der Version von Plug-ins oder auch Java verträgt, in diesem Fall kann Gradle auch manuell installiert werden. Sollte das Tool Update-Bedarf anmelden, empfiehlt es sich in der Regel, diesem Rat zu folgen.

### **Tipp: Update Gradle-Wrapper**

Falls sich Fehlermeldungen beim Erstellen des ersten Projekts zeigen, die anzeigen, dass Komponenten nicht gefunden werden oder der Gradle Sync nicht erfolgreich durchgeführt wird, empfiehlt sich ein Check der Version(en) von Java, IntelliJ, Kotlin-Plug-in für IntelliJ und Gradle-Wrapper. Sollte dieser nicht aktuell sein, ist ein Update möglich:

```
./gradlew wrapper --gradle-version 4.8.1 (aktuelle Version)
```

Mehr Informationen zu Gradle und Stand-Alone-Downloads gibt es unter folgender Adresse:

<https://gradle.org/>



Abb. 2.3: Positives Feedback vom gradle in einem leeren Projekt

Mit diesem leeren Projekt kann es im Prinzip losgehen.

Als Nächstes müssen noch grundlegende Verzeichnisse angelegt werden (oder die Option NEUE MODULE mit leerem Source-Verzeichnis anlegen und in den Settings anhängen.)

Die Ordnerstruktur sollte folgenden Aufbau haben: *src/main/kotlin*.

Nun ist es an der Zeit, darin die erste Datei anzulegen. Passend zum Projekt soll sie den Namen *firstrun.kt* tragen. Diese Datei benötigt ein wenig »overhead«, damit die Ausführung leicht möglich ist. Die Anweisungen machen die Kotlin-Anwendung kompatibel zur JVM. Das Ergebnis dieser Schritte sollte dann so aussehen:

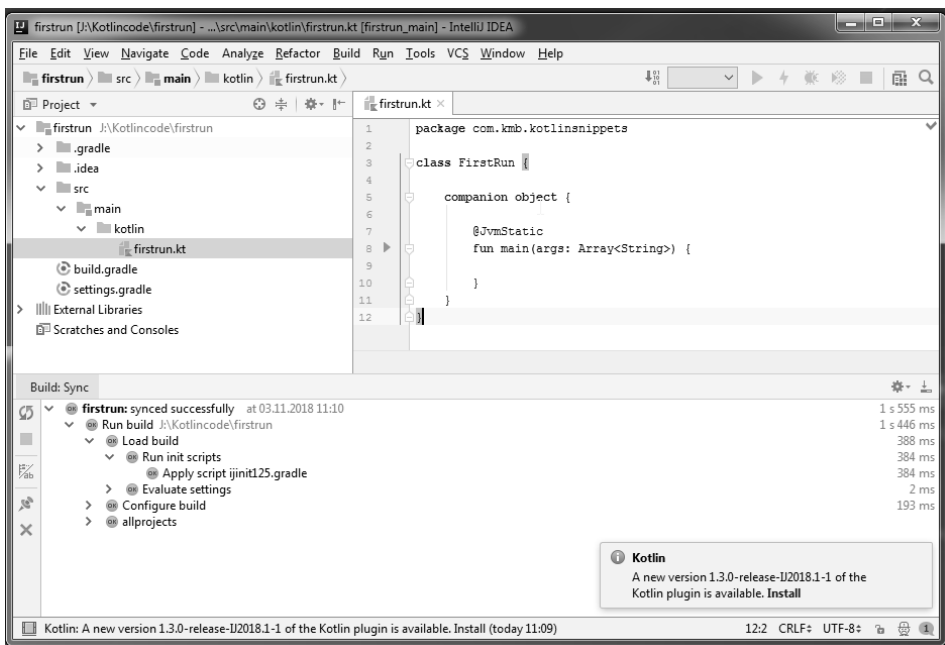


Abb. 2.4: Projektbildschirm mit Struktur (links), Code (rechts) und Build-Fenster (unten)

Der letzte Schritt ist dann, eine *Run-Configuration* zu erstellen, damit der Play-Button zum Ausprobieren des Codes glücken kann.

Dazu wählen Sie in dem leeren Dropdown-Menü neben dem ausgegrauten Play-Button (vorheriger Screen rechts oben) den Punkt EDIT CONFIGURATIONS ... oder KONFIGURATIONEN BEARBEITEN .... Dort folgt mit dem Klick auf das PLUS das Auswahlmenü für die Konfigurationstypen.

Hier wählen Sie KOTLIN aus, um die grundlegenden Einstellungen für das erste Projekt zu öffnen (s. Abbildung 2.5)

Bei diesem letzten Schritt müssen Sie einmal die gerade erzeugte Klasse in dem hervorgehobenen Bereich USE CLASSPATH OF SUBMODULE auswählen.

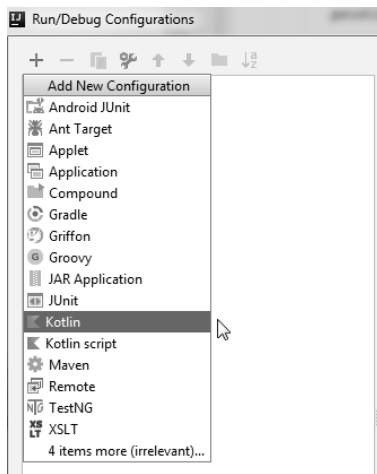


Abb. 2.5: Build-Konfigurationen

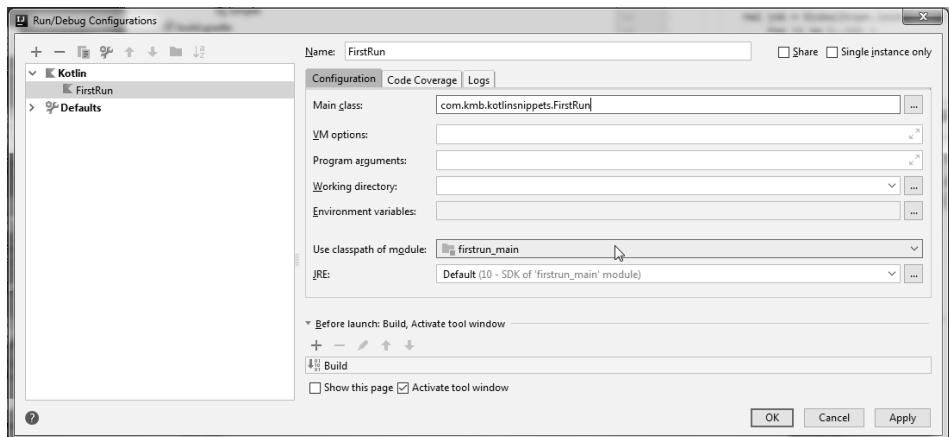
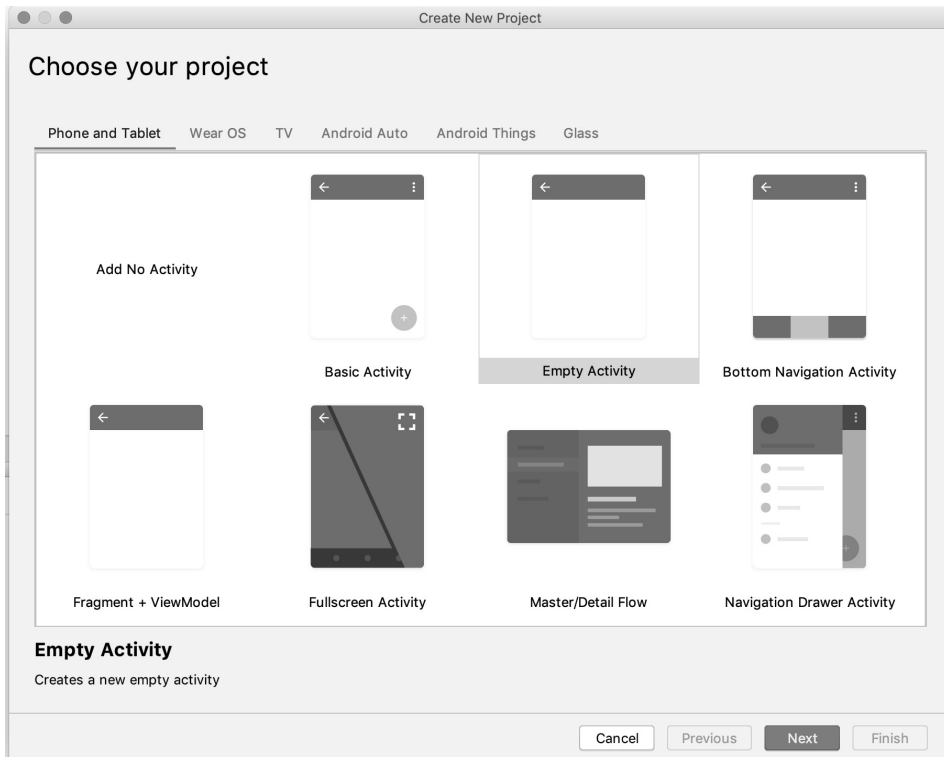


Abb. 2.6: Build-Konfigurationsdetails

Im folgenden Kapitel folgt Inhalt, mit dem die `main`-Methode gefüllt und ausprobiert werden kann.

## 2.1.2 Android Studio

Mit dem Projektassistenten legen Sie auch in Android Studio eine leere Applikation an. Dieser Schritt generiert bereits ersten Kotlin-Code und ein leeres Layout, in dem Sie direkt arbeiten können und das auf dem Simulator oder dem eigenen Handy einen App Screen erzeugen sollte.



**Abb. 2.7:** Projektassistent Android Studio

Im ersten Schritt scheint das Template `EMPTY ACTIVITY` zu passen, da es eine funktionierende Grundlage schafft, ohne viel Android-Infrastruktur zu erstellen, die in den ersten Schritten nicht notwendig ist. Wer sich mit Android auskennt, kann selbstverständlich auch mit einer `ANDROID TV APP` oder einer `FRAGMENT VIEW-MODEL`-Variante beginnen.

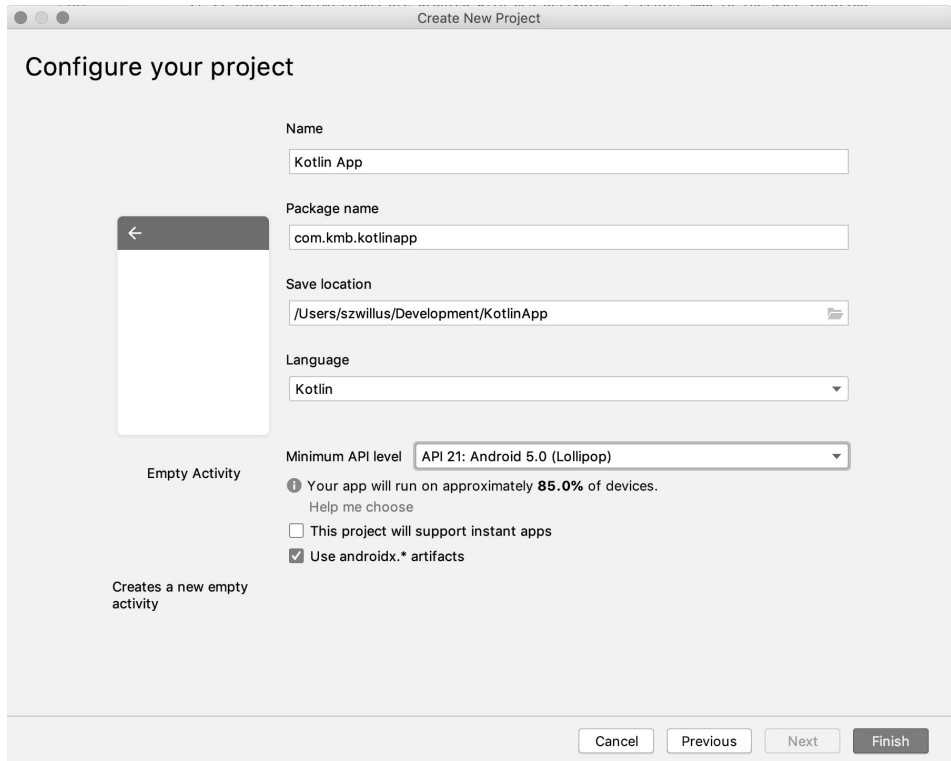


Abb. 2.8: Projektdetails konfigurieren

Im nächsten Schritt legen Sie die Details fest, eventuell fällt hier auch erneute Downloadzeit für die richtige Android-Version an. Im Beispiel habe ich mich für eine recht alte API-Level entschieden, in einem Testprojekt kann das gerne aktueller sein. Der Name der App kann frei gewählt werden, der Package-Name muss den Konventionen für Java-Packages folgen und darf nur Kleinbuchstaben ohne Sonderzeichen enthalten.

Das Resultat ist eine Minimalapp im Code, die auch direkt zum Ausführen konfiguriert ist.

Die ersten Beispiele in diesem Buch gehen davon aus, dass der Code auf der Konsole ausgeführt wird. Es ist aber kein Problem, die Beispiele leicht anzupassen und sie dann entweder aus der Konsole von Android zu sehen oder aber in einem TextView und damit auch direkt in einer App auszugeben.

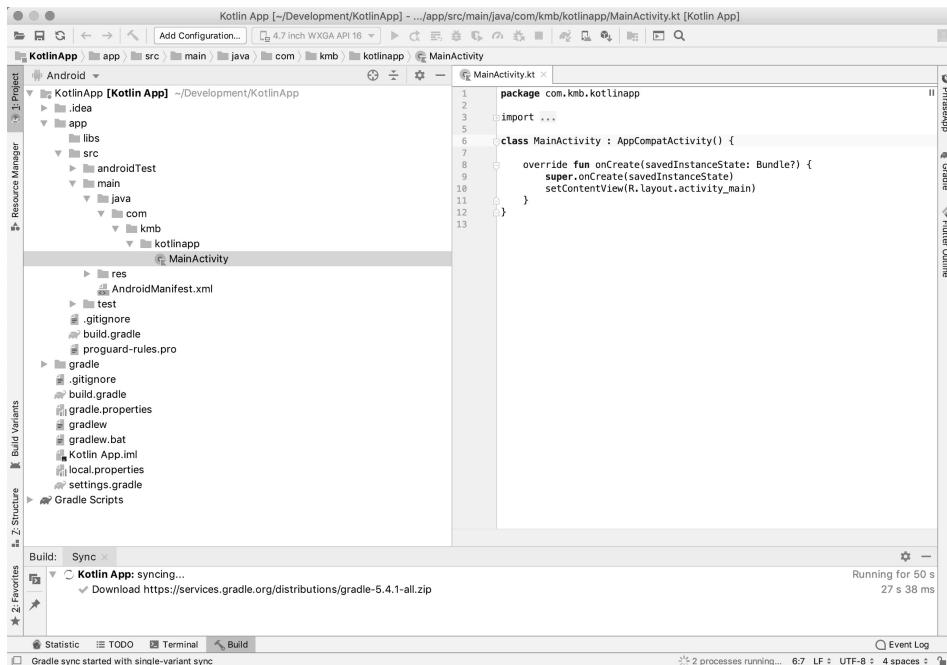


Abb. 2.9: Hauptfenster Android Studio mit Projektdaten

### Hinweis: Nutzung mit einem Android-Projekt

Um die einfache Konsolenausgabe zu erreichen, die in den Codebeispielen mit `print("Irgendwas")` gemacht wird, kann in einer Android-App `System.out.print("Irgendwas")` verwendet werden – bzw. die Variante mit `println`.

Um Ausgaben mit dem `TextView` in einer App zu machen, muss dieser einmal in der Applikation aus der Layout-Datei gelesen werden (oder mittels *kotlin-android-extensions* direkt importiert) und kann dann mit `textView.text = "Irgendwas"` genutzt werden, um die Ausgabe zu machen.

Führt man die App einmal aus und lässt sie sich anzeigen, so grüßen die Worte »Hello world« in einem ansonsten leeren App-Screen mit dem Titel *Kotlin App*.

## 2.2 Tools

Kotlin als Programmiersprache ist einigermaßen Tool-agnostisch. Sie können in einem beliebigen Text-Editor entwickeln. Größeren Komfort und sinnvolle Unterstützung bietet jedoch die integrierte Entwicklungsumgebung (IDE) IntelliJ oder das davon abgeleitete Android Studio.

Von meiner Seite gibt es daher auch die klare Empfehlung, mit der IntelliJ-Community-Edition für die ersten Beispiele zu arbeiten. Für Entwickler von Android-Apps gibt es wenig Gründe, nicht in Android Studio weiterzumachen und gegebenenfalls die entsprechende Kotlin-Unterstützung zu aktivieren.

Um kleine Code-Schnipsel einmal direkt in Aktion zu sehen, habe ich außerdem immer wieder auf die Online-Compiler unter <https://try.kotlinlang.org> zurückgegriffen.

### 2.2.1 Kotlin-Compiler

Die grundlegendste Möglichkeit, Kotlin zu nutzen, ist der Kotlin-Compiler auf der Kommandozeile. Der Compiler wird dabei verwendet, um den bestehenden Code mit der Kotlin-Runtime zu bündeln und dann in ein ausführbares JAR zu konvertieren, welches mit einer Java-Umgebung lauffähig ist.

Darüber hinaus kann der Kotlin-Compiler auch als interaktive Shell oder zum Scripting verwendet werden.

#### **Tipp**

Zum Umgang mit dem Kotlin-Compiler und seiner Installation:  
<https://kotlinlang.org/docs/tutorials/command-line.html>

Die ersten Code-Beispiele kann man theoretisch auf diese Art ausführen, es ist aber an sich nicht zu empfehlen, da es keinerlei Unterstützung durch eine Umgebung gibt und auch das Management von Bibliotheken manuellen Aufwand bedeutet.

### 2.2.2 Kotlin-Compiler mit Gradle

Auf der Konsole kann auch das Buildsystem Gradle die Arbeit an größeren Projekten vereinfachen. In beiden IDEs ist es dafür auch standardmäßig eingebunden. Gradle konfiguriert und managt die Abhängigkeiten eines Programms. So benötigt schon die einfachste »Hello world!«-Anwendung mindestens die Kotlin Runtime Library. Falls man unbedingt ohne eine Entwicklungsumgebung arbeiten wollte, wäre die Empfehlung, mindestens Gradle zusätzlich zu nutzen.

#### **Tipp Gradle lokal installieren**

Unter der domain <https://gradle.org/> gibt es die Gradle Build Tools als Stand-alone-Software. Als nächster Schritt helfen dann die Getting Started Guides: <https://gradle.org/guides/#getting-started>.



In diesem Fall wird eine Konfigurationsdatei namens `build.gradle` erstellt, in der die Einstellungen festgehalten werden. Damit kümmert sich das Build Tool dann um den Download von Bibliotheken und Frameworks und konfiguriert den Kotlin/Java-Compiler entsprechend der gewünschten Einstellungen.

### 2.2.3 Integrierte Entwicklungsumgebungen/IDE

Kotlin ist als Multiplattform-Projekt nicht auf ein Tool festgelegt, aber für eine große Zahl an Umsteigern dürften die Produkte von JetBrains eine bekannte Referenz sein. Der Support im Tool (IntelliJ ab Version 15, Android Studio ab Version 3.0) entspricht dem Standard, der aus der Java-Entwicklung bekannt ist.

Dazu gehören Syntax Highlighting, Code completion und automatische Imports ebenso wie die von IntelliJ bekannten Inspections und Analysetools für den Code.

Wie es der Name schon nahelegt, ist die Android-Studio IDE auf Android-Apps festgelegt, weitergehender Support, wie zum Beispiel Projekt-Templates für eine JS-Anwendung, fehlt hier. Umgekehrt ist das allgemeine IntelliJ-Bundle mit allen notwendigen Komponenten für die komplette Android-Entwicklung ausgestattet.

IntelliJ erlaubt auch die Entwicklung von Android-Anwendungen und zusätzlich auch die weiteren Verwendungen von Kotlin. Die Beispiele in diesem Buch sind größtenteils mit der IntelliJ-IDE und Android Studio entstanden. Beide IDEs sind unter bestimmten Bedingungen frei erhältlich.

#### **Tipp: Download URLs**

<http://www.jetbrains.com/idea/download/index.html>

<https://developer.android.com/studio/>

### 2.2.4 Konvertierung von existierendem Code

Eine sehr praktische Funktion – über beide IDE leicht zu erreichen – ist die Konvertierung von bestehendem Java-Quelltext in Kotlin. So kann man die neue Sprache von bekanntem Terrain her erobern. Bei der Umsetzung entsteht in der Regel nicht die ideale Kotlin-Lösung, aber im besten Fall schon mal kompilierender und funktionierender Code.

Doch es gibt ein paar Aspekte zu beachten, die auch nach der automatischen Code-Konvertierung noch notwendig sind. Zum einen tut sich die Konvertierung mit komplexen `if/switch`-Anweisungen schwer und erzeugt bisweilen duplizierten Code.

Zum anderen wird die Konvertierung alle Unsauberkeiten in Bezug auf den Umgang mit Null-Sicherheit oder Typen aufdecken und damit unter Umständen

auf relativ ungünstige bzw. teure Strukturen konvertieren, die besser manuell korrigiert werden. Zum Teil kam es hier bei meiner Arbeit an Projekten für das Buch auch zu Fehlern, die entweder nicht kompilierte oder aber Laufzeitfehler zeigten.

Last but not least entsteht auch im besten Fall Java-Code in Kotlin-Notierung. Die Paradigmen funktionaler Parameter werden also eher wenig bis gar nicht umgesetzt. Die Konvertierungsroutine erzeugt zum Beispiel veränderliche Variablen, die in Kotlin zum Beispiel besser als `immutable` deklariert werden sollten.

# Stichwortverzeichnis

## @

- Annotations 116
- Schleifen-Label 111

## A

- abstract 147
- abstrakte Klassen 147
- Activity 55
- Actual 333
- actual 143
- also 218
- Android 55, 265
  - Activity 23, 55, 271
  - Adapter-Views 281
  - Änderungen durch Kotlin 266
  - Application 271
  - architecture components 283
  - Architektur 283
  - Coroutinen 303
  - Delegation 304
  - Delegation-Basisklasse 306
  - Extension-Funktionen 265
  - Extension-Methode 305
  - Extension-Property 305
  - Fragment 271
  - Interactor 287
  - JetPack 271
  - Klassendelegation 306
  - Kotlin-Vorteile 265, 267
  - Layout-Dateien 55
  - Lifecycle 271, 282
  - Package-Struktur 266
  - Probleme 23
  - Remote API nutzen 300
  - Repository 287
  - Retrofit 300
  - Scope einer Coroutine 304
  - Vererbung reduzieren 268
  - Views 282
- Android-Extensions 59
- Android KTX 285
- Android Studio 33, 37, 41
  - Bytecode-Ansicht 142
  - Download 41
- Android-View
  - Konstruktoren überladen 315
- Annotations
  - benutzen 116
  - Parameter 117
  - Test 191, 203
  - Use-Site target 317
- anonyme Klasse 166, 168
- Anweisungen 63
- Anweisungsblock 64
- Any 73, 93, 148
- apply 218
- Architekturmodelle 24
- Arrays 47, 73, 94, 224, 225
  - copyOf 76
  - erstellen 226
  - erzeugen 61, 76
  - Factories 226
  - Initialisieren 94
  - Iteration 230
  - mehrdimensional 95, 122
  - nutzen 224
  - Skalararrays 95, 225
  - Standard-Konstruktor 227
  - transformieren 231
  - Zugriff 91, 228
- Arrays vs. Listen 224
- as 119, 120
- as? 119, 120
- Assertion 203
- async 258
- Attribute 145
- Ausdrücke 63
- Autoboxing 86
- Automatische Tests 47, 187
  - Assertion 203
  - Auswertung 200
  - Behaviour Driven Design 209
  - Beispiel 195
  - Codequalität 201
  - Integrationstests 187
  - Matcher 209
  - Mock 189
  - Multiplattform 211

- parametrisierte Tests 193
- RSpec 209
- Setup 206
- Strategien 206
- SystemOutRule 203
- Testabdeckung 200
- Testgestaltung 192
- Unit-Tests 187
- await 258

## B

- backing field 160
- Behaviour Driven Design 209
- Blackbox-Tests 188
- boole'scher Ausdruck 49
- Boolean 72, 90
- Boxing 81, 85
- break 111
- by 237
- Byte 72, 88
- Bytecode-Optimierung
  - inline 141

## C

- Call-Chaining 61
- capitalize 113, 234
- Casts 119
  - Smart cast 121
- catch 115
- Char 73, 90
- class 55, 65
- Class Delegation 241
- class members 146
- Clean Architecture 285
- Code
  - Architekturmodell 145
  - Strukturierung 145
- Code Coverage 201
- Code-Konvertierung 41
- Codequalität 201
- Collection 224
- Companion Object 58, 165
  - Methoden aufrufen 166
- Comparator 230
- componentN 134, 170, 185
- continue 111
- Contracts 258
  - callInPlace 260
  - returns 261
  - Smartcasts 260
  - Zusicherungen 259
- copyOf 76

## Coroutinen

- Actor 258
- Android 303
- async 258
- await 258
- experimentelle Klassen 258
- launch 253
- Scope 253, 255
- ScopeBuilder 255
- Suspend-Funktion 254

## Coroutines 253

- Custom Setter 239

## D

- data 60, 170
- Datenklasse 60, 170
- Datentyp
  - Char 90
  - Range 99
- Datentypen 72
  - ? 77
  - Array 47, 73, 94
  - Boolean 72, 90
  - Byte 72, 88
  - Casting 89
  - Char 73
  - Double 73, 88
  - dynamic 321
  - Fließkommazahlen 88
  - Float 73, 88
  - Funktionen 137
  - Ganzzahlen 88
  - Generics 85
  - Generische Typen 85
  - Int 47, 72, 88
  - konvertieren 118
  - Konvertierung 119, 120
  - List 53, 73
  - Liste 96
  - Long 88
  - Map 98
  - nullbar 49
  - Range 73
  - Set 96
  - Short 88
  - skalare 73
  - String 47, 73, 90
  - Zahlen 88
- Deferred 258
- dekompilierte Java-Klassen 143
- delay 255
- Delegation 236
  - Android 268, 269, 304

- Class Delegation 241, 243
- Custom 241
- Custom Property Delegation 305
- inject 251
- Klassendelegation 306
- lazy 237
- map 237, 241
- Observable 239
- observable 237
- property 237
- Property Delegation 304
- Vetoable 240
- vetoable 237
- Delegation vs. Vererbung 236, 241
- Dependency Injection 190, 246, 248, 250, 251
- Dependency Inversion 285
- Destrukturierende Deklarationen 184
- Destrukturierung von Deklarationen 133
- Document Object Model 323
- Dokumentation 218
  - Plattformen 30
- DOM 323
- Domain-Specific-Languages 187
- Double 73, 88
- do while 109
- Downloads 17, 18
- DSL 187

## E

- eager loading 252
- else 102
- Elvis-Operator 50
- Enums 179
  - ordinal 180
  - toString 180
  - valueOf 180
- Enums vs. Sealed Klasse 181
- Exception 48, 115
- Exception handling 114
- expected 143, 333
- Expressions 63
- Extension-Funktionen 26, 59, 182
  - Android 269, 302
  - für eigene Klassen 184
  - importieren 154
  - Vererbung 183

## F

- Factory 157
- finally 115
- first 229
- firstOrNull 228
- Float 73, 88

- fold 112
- folding 232
- for 106, 107, 108
- forEach 112
- foreach 107
- Fragezeichen-Operator 50
- fun 45, 127
- Funktion
  - Deklaration 45
  - Funktionskörper 67
  - Kurzschreibweisen 45
  - Parameter 45
- Funktionale Programmierung 22, 125
  - Beispiel 27
  - Höhere Funktionen 24
  - Paradigma 125
  - reine Funktion 24
  - Seiteneffekte 44
  - unveränderliche Datenstrukturen 25
- Funktionen 125
  - actual 143, 333
  - als Parameter 136
  - anonyme Funktion 137
  - Aufrufen 70
  - ComponentN 134
  - Definieren 127
  - einzeilig 129
  - expected 143, 333
  - first-class citizens 126
  - Funktionskopf 71, 128
  - generische Funktionen 177
  - höhere Funktionen 51
  - infix 143
  - inline 141, 178
  - Lambdas 126
  - lokale Funktionen 139
  - Methoden (in Klassen) 161
  - named parameter 130
  - Namen 127
  - Operator überladen 144
  - Parameter 127, 129
  - reified 178
  - rekursiv 138
  - return 128, 132
  - Rückgabotyp 127
  - Scope-Funktionen 218
  - Signatur 128
  - Top-Level 66
  - Unit 127
  - variable Parameter 131
  - verketteten 135
- Funktionsaufrufe 70

**G**

- Generics 85, 171
  - Beschränkung 173
  - invariante Typen 174
  - Klassendefinition 171
  - kontravariante Typen 176
  - kovariante Typen 175
  - mehrere Typen 172
  - Varianz 174
  - Vererbung 173
- generische Klassen 171
- generische Methoden 177
- generische Typen 171
- getOrElse 229
- getOrPut 229
- Getter 159
- Gherkin Cucumber Style 210
- Google I/O 13
- Gradle 34, 40
  - Versionskonflikt 34
  - Wrapper 34
- Gradle-Projekt
  - Android 272

**H**

- HamKrest 190
- höhere Funktionen 51, 136

**I**

- IDE
  - Android Studio 33, 41
  - IntelliJ 41
- idiomatische Ausdrücke 44
- if 49, 102
- if/else 102
- import 56, 152
- infix 143
- init 155
- Initializer 74, 156
- inject 251
- inline 141
- Inlining
  - Funktionen 141
  - Lambdas 141
- innere Klassen 163
- Int 47, 72, 88
- Integrationstests 187
- IntelliJ 13, 33, 41
  - BYTECODE-ANSICHT 142
  - Download 41
- Interactor 248
- Interface 147

- internal 69, 152
- Interoperabilität 22
- Inversion of Control 246
- isNotNullOrEmpty 234
- it 52
- Iterable 106

**J**

- Java 41
  - Annotation-Kompatibilität 317
  - Annotations 314
  - Autoboxing 86
  - Bibliotheken nutzen 312
  - char as int 89
  - Collections 314
  - Default-Parameter 315
  - Derivate 21
  - final 75
  - final static-Werte 165
  - funktionale Muster 126
  - implizites Casting 89
  - instanceof 121
  - Interoperabilität 311
  - JvmOverloads 315
  - Kompatibilität 22
  - Konvertieren 41
  - Kotlin-Kompatibilität 30
  - Multiplattform 312
  - Packages (Unterschiede) 149
  - Probleme 21, 127
  - Reflection 234, 312, 314
  - Sichtbarkeitsmodifizier 152
  - Sichtbarmachung 316
  - skalare Datentypen 73
  - statische Felder 316
  - Streams 314
  - switch/case 104
  - ternärer Operator 102
  - Threading 253
  - throws 115
  - überladene Funktionen 130
  - Wertecache 87
  - Zielformat 312
- Java-Annotations
  - JvmField 315
  - JvmOverloads 315
  - JvmStatic 315
- Java-Entwicklung 21
- Java-Kompatibilität 30
- JavaScript
  - Bibliotheken 311
  - Document 319, 323

- DOM 319, 323
- dynamic 321
- external 327
- Funktionen aufrufen 326
- IntelliJ-Projekt einrichten 318
- JavaScript-Runtime 318
- js 327
- JSON 324
- Kotlin aus JS aufrufen 319
- MVP 320
- Promise 253, 320
- statische Typen 29
- TypeScript 29, 311
- Unterschiede 29
- Java-Virtual-Machine 58
- JetBrains 13
- joinToString 233
- JS
  - Aufruf 327
- JUnit 190, 191
  - After 207
  - AfterClass 207
  - assertFalse 192
  - assertTrue 192
  - Before 207
  - BeforeClass 207
  - parametrisierte Tests 207
- JUnit 4 191
- JUnit 5 192
- JVM 58
- K**
- KClass 235
- KFunction 235
- Klassen 55
  - abstrakte 147
  - Actual 333
  - anonyme Klasse 166, 168
  - anonyme Klassen (Rückgabe) 169
  - Attribute 145
  - class member 65
  - companion object 157, 165
  - ComponentN 170, 185
  - Custom Getter/Setter 159
  - Datenklassen 59, 170
  - Deklaration 65
  - Eigenschaften 146
  - enum 179
  - expected 333
  - Extensions 182
  - Factory 165
  - Importe und Packaging 154
  - importieren 152
  - init 155
  - Initialisierung 155
  - Initializer 156
  - Initializer vs. Custom Getter 161
  - inner 163
  - innere Klassen 163
  - Interfaces 147
  - Konstruktor 61, 154
  - member 146, 154
  - Member-Funktionen 71
  - member überschreiben 148
  - Methoden 145, 161
  - nested 163
  - open 147
  - primäre und sekundäre Konstruktoren 155
  - privater Konstruktor 157
  - Properties 60, 145, 146, 148, 151, 158, 182, 185
  - Properties deklarieren 158
  - Properties-Sichtbarkeit 158
  - Property-Setter 159
  - Schachtelung 163
  - sealed 181
  - Sichtbarkeitsmodifizier 151
  - Strukturelement 146
  - überschreiben 148
  - Unit-Tests 204
  - variable Properties 158
  - veränderliche Properties 82
  - Vererbung 146
- KOIN 246, 251
  - factory 252
  - single 252
- Konstruktor 61
- Kontrollstrukturen 101
- Kotlin
  - Code-Sharing 338
- kotlin2js 327
- kotlin.test 190, 212
- Kotlin/Native 338
- Kotlin-Android-Extensions
  - Layout import 280
- Kotlintest.io 210
  - TestCases 211
- Kurzschreibweisen
  - Funktionen 129
  - it 52, 72
  - Lambdas 68, 71
  - Type Aliases 122

**L**

- Lambdas 50, 126
  - Chaining 138
  - it 72
  - Parameter 72
  - Rückgabewert 72, 138
  - Verketteten 138
- lazy 237
- lazy-Initialisierung 238
- let 50, 218
- List 73, 225
- Listen 96
  - erzeugen 61
  - Factories 226
  - fold 232
  - Iteration 230
  - nutzen 224
  - transformieren 231
  - Zugriff 228
- lokale Funktionen 139
- Long 88

**M**

- map 112, 113, 237
- Map-Delegate 241
- mapIndexed 112
- Mapping 26
- Maps 98
  - erzeugen 227
  - Factories 227
  - getOrPut 229
  - iterieren 231
  - Zugriff 229
- Matchers 209
- mehrdimensionale Arrays 95
- Methoden
  - abstract 147
  - aufrufen 162
  - definieren 161
  - open 147
  - Sichtbarkeit 162
- Mock 189, 190, 247
- Mocking 189
- Mockito 190
- Model-View-Presenter 285, 286
- Model-View-ViewModel 285
- Multiplattform 22
  - iOS 29
  - JavaScript 29
  - JVM 29
  - plattformspezifisch 30
- MutableList 225

- MutableMaps 225
- MutableSets 225
- MVP 247, 285
  - Activity (View) 289
  - Datenupdates 293
  - Initialisierung 288
  - Kritik 296
  - Nutzereingaben 293
  - Presenter 287, 288
  - Repository 291
  - Testbarkeit 291
  - Vertrag (Contract) 287
  - View 287
  - ViewModel 287
  - Vorteile 286
- MVP vs. MVVM 302
- MVVM 247, 285, 297
  - Daten laden 298
  - Initialisierung 297
  - LiveData 298
  - MutableLiveData 298
  - Observer 297
  - ViewModel 297
  - Vorteile 296
- MVVM-Anwendung 297

**N**

- named parameter 130
- Nebenläufigkeit 253
- nested class 163
- non-null-assertion 78
- null 46, 49
- Null-Ausschluss 78
- Null-Pointer 79
- Null-Sicherheit 77, 83

**O**

- object 58, 165
- Objekt-Anweisungen 166
- Objekt-Ausdrücke 166
- Objekte 55, 93
  - Konstruktor 93
- Objektorientierung 55, 145
  - Beispiel 26
  - Klassen 146
  - Methoden 162
  - Probleme 23
  - Seiteneffekte 26
  - Vererbung 56, 146
  - Vorteile 28
- observable 237, 239
- open 147



operator 144

Operatoren

:: 234

!! 78

? 50, 78, 80, 84

\* 131

\$ 61, 92

as 119, 120

as? 119, 120

downTo 99

in 91, 176

is 121

out 175

step 99

to 227

until 99

org.w3c.dom 323

override 148

## P

Packages 56, 149

Grundpackage 149

gruppieren 149

import 152

internal 152

Sub-Packages 149

Use-Cases 150

Parameter

Default-Wert 61, 130

parametrisierte Tests 207

Plattformen

Android 215

JS 215

JVM 215

primärer Konstruktor 155

println 45

private 69, 152

Projekte

Android 37

anlegen 33, 37

ausführen 36

bearbeiten 35

konfigurieren 34

Template 37

Properties 60

backing fields 160

Getter 159

Setter 159

Property Delegation 237, 304

protected 152

public 69, 151

## Q

QUnit 190

## R

Range 73, 107

Ranges 99

React Helper 326

ReadOnlyProperty 241

ReadWriteProperty 241

Receiver 71

Refactoring 197

Reflection 234

Class Members 235

KClass 235

KFunction 235

lateinit 235

Properties 235

regular Expression 91

reified 178

Rekursion 106, 138

rekursive Funktion 138

repeat 110

return 132

return early 132

reverse 234

reversed 113

RSpec 209

run 218

runBlocking 255

## S

SAM-Klassen 127, 168

Scala 21

Schleifen 53, 105

abbrechen 111

break 111

continue 111

do while 109

for 53, 106, 107

foreach 53, 107

funktionsbasiert 112

Range (for) 107

repeat 110

while 68, 109

Scope 69

Scope-Funktionen 50

also 218

apply 218

Benutzung 219

let 218

Parameter 219

Rückgabewert 219

- run 218
  - Schachtelungen 223
  - Transformation 220
  - Sealed 181
  - Sealed classes 181
  - Sealed Klasse vs. Enum 181
  - sekundäre Konstruktoren 155
  - selbstreflektiver Code 234
  - Semikolons 14
  - Sequence 224
  - Service Location 246
  - Service Locator 247, 249
  - Set 225
  - Sets 96
    - unique constraint 226
  - Setter 159
  - Shared Preferences
    - Delegation 305
  - Short 88
  - Sichtbarkeit
    - einschränken 151
    - internal 152
    - Lokale Variablen 152
    - private 152
    - protected 152
    - public 151
  - Sichtbarkeitsmodifizier 151
  - Sichtbarkeitsmodifikatoren
    - internal 69
    - private 69
    - public 69
  - Sieb des Erathostenes 108
  - Singleton 58, 157, 165, 166
  - Skalararray 95, 225
  - skalare Datentypen 73
  - slice 230
  - Smart Cast 121
  - Specification Style 210
  - Spek 210
    - group 210
    - test 210
  - Spracheigenschaften 22
    - concise 14
    - funktional 24, 44
    - Interoperabilität 29, 30
    - interoperable 14
    - Null-Sicherheit 24, 28, 46, 50, 58
    - objektorientiert 44
    - safe 14
    - Sicherheit 25, 28
    - statische Typisierung 24, 28, 118
  - Standardbibliothek 215, 216
    - alle Plattformen 216
    - plattformsspezifische Features 216, 217
  - Standardklassen
    - importieren 154
  - Statements 63
  - statisch typisiert 77
  - statisch typisierte Sprache 118
  - String 47, 73, 90
  - Strings
    - Escape-Sequenzen 92
    - Raw-String Format 92
    - String-Templates 61
    - Templates 92
    - Tools 233
    - trimMargin 93
  - String Templates 92
  - Struktur 65
  - Stub 247
  - sublist 230
  - suspend 254
  - Syntax 63
- T**
- takeif 223
  - takeunless 223
  - Testabdeckung 189, 201
  - Test-Annotation 203
  - Testbarkeit 26, 187
    - Beispiel 195
  - Test Driven Development 187
  - Testing
    - DSL 208
    - Vorteile 198
  - Testprojekt 193
  - Tests 187
  - Test-Stile 209
  - Testsuite 189
  - Texttools 233
  - toInt 118
  - toLowerCase 113
  - Tools 39
    - Kotlin-Compiler 40
  - Top-Level-Funktion 66, 149
    - Android 268
  - Top-Level-Properties 149
  - Transformation 231
  - try 115
  - try/catch 114
  - Türme von Hanoi-Algorithmus 139
  - Type Aliases 122
  - Type-Casts
    - Konvertierung 119
  - Typinferenz 50, 72, 84
  - Typsicherheit 77

**U**

- Unicode 91
- Unit 45
- Unit-Tests 47, 187
  - Assert 191
  - assertEquals 191
- unsafe Cast operator 120
- use 314

**V**

- val 49, 74
- var 49, 74
- vararg 131
- Variable
  - null-safe 58
- Variablen
  - Boxing 81
  - Deklaration 74
  - Identität 86
  - immutable 74
  - initialisieren 74
  - mutable 74
  - nullbar 77
  - nullbar vs. nicht-nullbar 81
  - Referenz 74
  - Scope 69
  - Sichtbarkeit 67
  - transformieren 77
  - Typinferenz 84
  - unveränderlich 49, 74
  - veränderlich 49, 74

- Wert 74

- Zugriff 75

- Zuweisung 64

- Variablenscope 69

- Vererbung 56

- Vererbungshierarchie 146

- Vererbung vs. Delegation 236, 241

- Verkettung von Null-Checks 83

- Verzweigungen 49

- if 101

- when 102, 103

- vetoable 237, 240

- Vorteile 26

**W**

- when

- Ausdruck 104

- enum 105

- Range 104

- sealed class 105

- while 68, 109

- White-Box Test 188

- withContext 256

- withIndex 231

**Z**

- Zahlen 88

- konvertieren 47

- Zielpattformen 216

- Zuweisung 64