Kotlin Crash Course

Fast-track your programming skills with practical experience

Elena van Engelen - Maslova

www.bpbonline.com

First Edition 2024

Copyright © BPB Publications, India

ISBN: 978-93-55516-305

LIMITS OF LIABILITY AND DISCLAIMER OF WARRANTY

Dedicated to

My father ***Victor*** *(R.I.P.)*

About the Author

Elena is an expert software engineer with 24 years of experience in software engineering. She holds a BSc in Computing and an MSc in Software Engineering. She specializes in Kotlin, Event Driven Architecture, Cloud native applications, Serverless cloud, microservices, DevOps, and Infrastructure as Code. She has extensive experience in cloud platforms, including AWS, Google Cloud, and Azure, and holds various certifications for AWS and Azure. Currently, she works as a Kotlin Tech Lead. Embracing the "we build it, we run it" philosophy, she strongly believes in DevOps. As a proud mother of two, she balances her demanding career with the joys and responsibilities of family life.

About the Reviewer

Nakul Pandey, as a PMTS (Principal Member of Technical Staff) at Salesforce and a former tech lead at Google, brings over 15 years of profound expertise in developing scalable mobile solutions. His technical journey includes pivotal contributions to resource management systems at Google, optimizing their internal infrastructure to meet strategic business goals. Currently, at Salesforce, he leads innovative projects in the mobile app development realm. His deep understanding of modern mobile technologies and his leadership in driving project delivery make him a respected figure in the mobile development community.

Acknowledgement

I would like to express my heartfelt gratitude to my father, who has always been my role model and source of inspiration. A distinguished figure in mathematical physics, he encouraged me to pursue a career in IT, recognizing that my passion for science would be fulfilled in this ever-evolving field.

I also want to acknowledge the developers at AZL, who I had the privilege of training. Their feedback and dedication played a crucial role in shaping the contents of this book.

Lastly, I extend my thanks to my family for their patience and understanding, allowing me the time and space to complete this project.

Preface

Kotlin Crash Course is a fast-paced, hands-on introduction to Kotlin, preparing readers to build robust applications efficiently using the latest language features and best practices.

The book is divided into comprehensive chapters that cover key Kotlin programming topics such as object-oriented and functional programming, collections, concurrency, and unit testing. Each chapter takes a Learning by Doing approach, focusing on practical projects rather than solely theoretical knowledge. This strategy improves knowledge retention by simulating real-life experiences, allowing students to apply concepts in practice as they learn them.

Furthermore, the book is interwoven with an abundance of best practices obtained from industry experience. This approach ensures that even beginners can gain seasoned insights and apply their knowledge confidently to real-world challenges.

Whether you are new to Kotlin or looking to refresh your skills, this book offers a step-by-step guide to quickly mastering the language features, preparing you for advanced development tasks and boosting your career opportunities.

Chapter 1: Discovering the Power of Kotlin Programming - This chapter provides an introduction to Kotlin, a modern programming language that runs on the **Java Virtual Machine** (**JVM**), can compile to JavaScript, and can be used for native code via LLVM based backend or GraalVM Native. The chapter starts by explaining what Kotlin is and how it was developed, including its key features and benefits compared to other programming languages. It then dives into the advantages of using Kotlin, such as its concise syntax, null safety, interoperability with Java, and support for functional programming. The chapter also discusses the growing popularity of Kotlin in the industry and its potential for future development. Additionally, we will explain the "Learn by Doing" approach of this book. By the end of this chapter, readers will have a good understanding of Kotlin's advantages, its multi-platform capabilities, and why it is an attractive choice for modern software development, including frontend, backend, serverless, and cloud computing. In addition, readers will become aware of the book's very hands-on approach to learning Kotlin.

Chapter 2: Kotlin Syntax and Basic Coding Conventions - This chapter provides an overview of Kotlin's syntax and basic coding conventions. The chapter covers essential topics such as naming conventions, comments and documentation, and code formatting and style guides. Readers will learn the basic building blocks of the Kotlin language, including variables, functions, control flow statements, and null safety. The chapter also discusses how to write clean and maintainable code by adhering to Kotlin's coding conventions, including naming conventions and commenting guidelines. By the end of this chapter, readers will have a good understanding of Kotlin's syntax and coding conventions, allowing them to write clean and maintainable code in Kotlin. It also lays the foundation for more advanced programming concepts covered in later chapters.

Chapter 3: Setting up the Development Environment - This chapter guides readers through setting up their development environment with IntelliJ IDEA, a popular and user-friendly IDE for Kotlin development. The chapter covers how to install and configure IntelliJ IDEA, as well as how to create a Kotlin project and set up Gradle with Kotlin DSL for building and testing the application. This also includes building a simple Hello World like program using the new development environment, compiling and running the program, and generating KDoc documentation for it. By the end of this chapter, readers will have a fully functional Kotlin development environment set up and ready for building and testing Kotlin applications.

Chapter 4: Fundamental Building Blocks of Kotlin - This chapter covers the fundamental building blocks of Kotlin programming. Topics include variables and data types, string templates, functions, null safety, control flow statements, and exception handling. Through a dynamic, hands-on project, readers will progressively build a command-line Formula One simulation, applying new concepts in real-time. This chapter emphasizes practical application, encouraging experimentation with Kotlin in IntelliJ's scratch files. Readers will learn how to declare variables, create functions, and use control flow statements to create conditional statements and loops. The chapter also covers null safety, an important feature in Kotlin that helps prevent null pointer exceptions. By the end of this chapter, readers will have a solid understanding of the basics of Kotlin programming, which will be essential for the more advanced programming concepts covered in later chapters.

Chapter 5: Object-oriented Programming - This chapter covers the fundamental principles of Object-oriented Programming (OOP) in Kotlin, including classes and objects, inheritance, polymorphism, interfaces, and generics. The chapter also covers advanced topics such as data classes, enum classes, sealed classes, and the object keyword. Readers will learn how to use smart casts and destructuring declarations to write clean and efficient Kotlin code. The chapter includes a hands-on project that provides practical experience in implementing OOP concepts. By the end of this chapter, readers will have a solid understanding of OOP principles in Kotlin and how to apply them in practice, enabling them to build robust and maintainable applications.

Chapter 6: Kotlin Collection Framework - This chapter covers Kotlin's powerful collection framework, which includes List, Set, and Map data structures. The chapter explains the key features of each collection type and explores how to manipulate their elements using various functions such as filter, map, any, all, none, find, findLast, first, last, count, associateBy, groupBy, partition, flatMap, minOrNull, maxOrNull, sorted, map element access, and zip. Readers will learn how to use these functions to perform common operations such as searching, filtering, and sorting data. The chapter also includes a hands-on project to help readers reinforce their understanding of the material. By the end of this chapter, readers will have a solid grasp of Kotlin's collection framework and will be able to use it effectively in their own code.

Chapter 7: Scope Functions - This chapter introduces readers to Kotlin's scope functions, including let, run, with, apply, and also. These functions provide a concise and efficient way to work with objects and simplify code. Readers will learn how to use each of the scope functions and when to apply them to different scenarios. The chapter includes a hands-on project to help readers practice using the scope functions in real-world examples, such as manipulating a list of integers and implementing a "safe call with let" operator. By the end of this chapter, readers will have a good understanding of Kotlin's scope functions and how they can improve their code.

Chapter 8: Functional Programming - This chapter dives into the world of functional programming, introducing readers to the key concepts and techniques for writing efficient and concise code. The chapter begins with an introduction to functional programming and the benefits it offers, followed by an exploration of higher-order functions, lambda functions, extension functions and properties, infix functions, operator overloading, and function composition. Readers will also learn how to work with Kotlin DSL using functional programming techniques, and will have the opportunity to apply their knowledge through a hands-on project that also includes a front-end component using Jetpack Compose.

Chapter 9: Exploring Delegation Design Pattern - This chapter explores delegation, a design pattern that allows objects to delegate certain responsibilities to other objects. In Kotlin, delegation can be achieved through the use of interfaces or delegated properties, which can lead to more efficient and concise code. By delegating responsibilities to other objects, the programmer can reduce code duplication, increase code reuse, and write more maintainable code. Additionally, delegated properties can be used to optimize object creation and memory usage. This chapter covers the delegation pattern, functional delegation, property delegation, and provides hands-on project that uses delegation within the business logic as well as Jetpack Compose frontend. By the end of this chapter, readers will have a solid understanding of delegation and its benefits for both the efficiency of code and the productivity of programmers.

Chapter 10: Concurrency and Parallelism - This chapter focuses on concurrency and parallelism in Kotlin, introducing both threads and coroutines as approaches to manage parallel execution. The chapter covers the basics of coroutines, structured concurrency, coroutines context and dispatcher, and context management to help readers write efficient concurrent programs. The topics of synchronization are also covered to ensure thread safety and avoid race conditions. Channels and flow are introduced as powerful tools for communication and data flow between coroutines, while exception handling is discussed to help manage errors in concurrent programs. Hands-on project allows readers to apply their knowledge in practice. By the end of this chapter, readers will have a solid understanding of concurrency and parallelism in Kotlin and be able to write efficient and safe concurrent programs.

Chapter 11: Unit Testing in Kotlin - This chapter introduces readers to unit testing in Kotlin, a critical aspect of modern software development. The chapter covers the basics of writing unit tests with Kotlin, including popular testing frameworks. Readers will learn hands-on how to write tests and how to use the MockK library to write tests for classes that depend on external services. The chapter also covers how to measure test coverage. By the end of this chapter, readers will have a solid understanding and hands-on experience in writing effective unit tests for their Kotlin code, ensuring that their code is reliable, maintainable, and robust.

Chapter 12: Building a Simple REST API- This chapter introduces readers to building a simple REST API with Kotlin using the Ktor framework. The chapter covers the basics of building endpoints and handling requests. By the end of the chapter, readers will have a fully functional REST API that can perform CRUD operations on a collection of data. The chapter also includes a hands-on project to help readers solidify their understanding of the material, including building the REST API, creating an integration test, and running the API locally, and trying it out using Swagger. Overall, this chapter provides a solid foundation for readers to start building their own REST APIs using Kotlin and Ktor.

Chapter 13: Building Event - Driven Cloud Native Serverless Application- In this chapter, the readers will learn about serverless computing, with a focus on its scalability and the simplicity of not having to manage infrastructure. Building a serverless application and deploying it to the cloud will provide readers with practical experience. In the process of building a serverless application, readers will learn about the benefits and challenges of serverless systems. Throughout the process, readers will experience how they can use Kotlin to write infrastructure as code to configure their cloud resources, how to implement the function as a service, and then automate their deployment to the cloud. The objective is to provide readers with a thorough grasp of how serverless technologies work in practice and to show why Kotlin is a good fit for modern cloud-based applications.

Code Bundle and Coloured Images

Please follow the link to download the
Code Bundle and the ***Coloured Images*** of the book:

https://rebrand.ly/5jheaco

The code bundle for the book is also hosted on GitHub at
https://github.com/bpbpublications/Kotlin-Crash-Course.
In case there's an update to the code, it will be updated on the existing GitHub repository.

We have code bundles from our rich catalogue of books and videos available at **https://github.com/bpbpublications**. Check them out!

Errata

We take immense pride in our work at BPB Publications and follow best practices to ensure the accuracy of our content to provide with an indulging reading experience to our subscribers. Our readers are our mirrors, and we use their inputs to reflect and improve upon human errors, if any, that may have occurred during the publishing processes involved. To let us maintain the quality and help us reach out to any readers who might be having difficulties due to any unforeseen errors, please write to us at :

errata@bpbonline.com

Your support, suggestions and feedbacks are highly appreciated by the BPB Publications' Family.

Join our book's Discord space

Join the book's Discord Workspace for Latest updates, Offers, Tech happenings around the world, New Release and Sessions with the Authors:

https://discord.bpbonline.com

Table of Contents

1. Discovering the Power of Kotlin Programming 1
Introduction........ 1
Structure........ 1
Objectives 2
Kotlin........ 2
Advantages of using Kotlin 3
All-around language capabilities 3
Object-oriented programming in Kotlin 3
Functional programming in Kotlin........ 3
Safety features 4
Multiplatform development 5
Interoperability with Java........ 5
Domain-Specific Languages........ 6
Data science........ 7
Popularity and community 8
Learn by doing approach of this book 8
Project preview........ 9
Conclusion........ 10

2. Kotlin Syntax and Basic Coding Conventions........ 11
Introduction........ 11
Structure........ 11
Objectives 12
Tooling and environment setup 12
Kotlin 1.9........ 12
Kotlin syntax overview........ 13
Kotlin source files 13
Kotlin visibility modifiers........ 14
Package definition........ 16
Import declarations 16
Program entry point........ 17
Variables and constants........ 17

Exercise: calculate the size of a shipping box ... 20
Functions ... 21
Null safety ... 24
Control structures ... 25
Defining and using classes ... 27
Annotations ... 29
Type inference ... 30
Error handling and exception management ... 31
Exception handling with try-catch-finally ... 32
Interoperability with Java ... 32
Naming conventions and code organization ... 33
Package names and directory structure ... 33
Source file names and organisation ... 35
Variables and properties ... 36
Constants ... 37
Functions, methods, and extension functions ... 38
Classes, interfaces, objects, and type aliases ... 39
Type parameters ... 41
Modifiers ... 42
Comments and documentation ... 43
Comment syntax and style guide ... 43
KDoc and documentation comments ... 44
Code formatting and style guide ... 46
Kotlin for mobile and desktop development ... 54
Cross-platform development with Kotlin ... 54
Styling and naming conventions across platforms ... 54
Community and Kotlin ecosystem ... 55
Conclusion ... 55

3. Setting up the Development Environment ... 57
Introduction ... 57
Structure ... 58
Objectives ... 58
Choosing the right tools for Kotlin development ... 58
Development environments for Kotlin ... 59

Gradle with Kotlin DSL ... 60
Setting up IntelliJ IDEA ... 60
Formatting and style ... 61
Enhancing developer experience ... 63
Creating a Kotlin Project ... 64
Create a new Kotlin Project ... 64
Inspect project structure ... 65
Setting Up Gradle with Kotlin DSL ... 67
Inspect build.gradle.kts ... 68
Setting JVM Target to JDK 21 ... 69
Writing your first Kotlin Program ... 70
Create a simple program ... 70
Run the program ... 71
Implement ComplimentGenerator ... 73
Modify program to use Compliment Generator ... 73
Insert a log line ... 74
Add KDoc ... 76
Compliment Generator: Full program ... 77
Conclusion ... 79

4. Fundamental Building Blocks of Kotlin ... 81
Introduction ... 81
Structure ... 81
Objectives ... 82
Variables ... 82
Mutable variables ... 82
Immutable variables ... 82
Variable scope and lifetime ... 83
Hands-on: use variables in your F1 app ... 84
Progress summary: variables in practice ... 88
Constants ... 88
Declaring constants ... 88
Restrictions on constants ... 89
Benefits of using constants ... 89
Hands-on: use constants in your F1 app ... 90

Progress summary: implementing constants *90*
Data types 90
Basic data types *91*
Arithmetic operations and numeric types 92
Handling arithmetic overflows *93*
Special types *94*
Collections *95*
Custom data types *96*
Hands-on: use data types in your F1 app *98*
Progress summary: Understanding data types *104*
String templates 104
String concatenation versus string templates *105*
String interpolation *105*
String templates with raw strings *105*
Advanced string interpolation and multiline handling *106*
String templates best practices *107*
Hands on: use String templates in your F1 app *107*
Progress summary: Race reports with string templates *109*
Functions 110
Top-level functions *110*
Member functions *110*
Nested functions *111*
Default arguments *112*
Named arguments *112*
Extension functions *113*
Lambda functions and higher-order functions *114*
Hands-on: use functions in your F1 app *115*
Progress summary: Utilized various type of functions *120*
Null safety 120
Nullability *121*
Safe call operator (?.) *121*
Elvis operator (?:) *122*
NotNull assertion operator (!!) *123*
Safe cast operator (as?) *124*
Advanced null safety techniques *124*

Common pitfalls in null handling ... *125*
Hands-on: use null safety features in your F1 app ... *125*
Progress summary: Incorporate null safety ... *127*
Control flow statements ... 128
Conditionals: if-else ... *128*
When expression ... *129*
For loops ... *131*
While loops ... *132*
Returns and jumps ... *133*
Hands-on: use control flow statements in your F1 app ... *134*
Progress summary: Utilizing control flow statements ... *137*
Exception handling ... 138
Exception basics ... *138*
Unchecked exceptions in Kotlin ... *138*
Throwing exceptions and creating custom exceptions ... *138*
Try-catch-finally ... *139*
Multiple catch blocks ... *139*
Alternative error handling strategy ... *139*
Exception handling best practices ... *140*
Hands-on: use exception handlingin your F1 app ... *141*
Progress summary: Utilizing control flow statements ... *144*
Conclusion ... 144

5. Object-oriented Programming ... 145
Introduction ... 145
Structure ... 145
Objectives ... 146
Classes and objects ... 146
Constructors ... *146*
Class relationships ... *147*
Understanding object references and identity ... *149*
lateinit properties ... *150*
Property accessors and backing properties ... *151*
Hands-on: create a DentalClinic app ... *152*
Introduction to Spring Boot ... *152*

Project setup with IntelliJ Ultimate *153*
Project Setup with IntelliJ Community Edition *154*
Checking gradle set up *154*
Building the object model *155*
Data Classes 158
Immutability in data classes *159*
Auto-generated functions *159*
Customizing auto-generated functions *160*
Serialization of data classes *161*
Hands-on: add data class 161
Enum classes 162
Properties and methods in Enum classes *162*
Enum Constants *162*
Enums and Singleton behavior *163*
Enum classes with abstract methods *163*
Hands-on: add enum *163*
Object keyword 164
Singleton pattern using the `object` keyword *164*
Companion objects *166*
Object expressions *167*
Hands-on: use the 'object' keyword *167*
Inheritance 168
Superclasses and Subclasses *168*
The 'open' keyword *169*
Overriding methods and properties *169*
The 'super' keyword *170*
The use of constructors in inheritance *171*
Hands-on: add inheritance *171*
Interfaces 173
Defining an interface *173*
Implementing interfaces *173*
Interfaces vs abstract classes *174*
Multiple interfaces implementation *174*
Default Interface methods *175*
Hands-on: add interfaces *175*

Polymorphism 179
Subtype Polymorphism *179*
Method Overriding *179*
Polymorphism with Interfaces *180*
Safe Casts and the 'is' keyword *182*
Hands-on: add polymorphism *182*
Generics 186
Generic classes and interfaces *186*
Type parameters *187*
Type constraints *187*
Reified type parameters and inline functions *187*
Hands-on: using generics *188*
Sealed classes 191
Defining sealed classes *192*
Advantages of sealed classes and sealed interfaces *193*
Sealed classes/interfaces vs. enum classes *194*
Hands-on: use sealed classes *195*
Value classes 200
Defining value classes 200
Using value classes *201*
Value classes vs regular classes and aliases *202*
Hands-on: use value classes *202*
Smart casts 204
Working with smart casts *204*
Null safety and smart casts *205*
Limitations of smart casts *206*
Hands-on: using smart cast *206*
Destructuring declarations 207
Destructuring declarations in variable assignments *208*
Destructuring declarations in loops *208*
Destructuring declarations with lambdas *208*
Limitations of destructuring declarations *208*
Hands-on: use destructuring declarations *209*
Annotations 211
Built-in annotations in Kotlin *211*

Creating custom annotations in Kotlin 212
Annotation targets and use-site targets in Kotlin 213
Hands-on: create a custom annotation for validation 214
Conclusion 216

6. Kotlin Collection Framework 217
Introduction 217
Structure 218
Objectives 218
Collections overview 219
Immutable collections 219
Mutable collections 220
List 221
Creating Lists 221
Performance implications 222
Accessing elements 222
Concatenating lists and adding elements 222
Adding and removing elements (Mutable Lists) 223
Common List operations 223
List equality and identity 223
Hands-on: implement a product catalog repository 223
Set 227
Creating Sets 227
Performance implications 227
Accessing elements 228
Adding and removing (MutableSet) 228
Checking for elements 228
Set operations 228
Conversion to and from Lists 229
Set equality and identity 229
Hands-on: use Set 229
Map 231
Creating Maps 231
Performance implications 231
Accessing values 232

Adding and removing entries (Mutable Maps) 232
Checking for keys and values 232
Map iteration 232
Map equality and identity 232
Hands-on: Implement a product inventory repository 233
Ranges and progressions 236
Closed-ended Range 236
Open-ended Range 236
Progressions 236
Hands-on: use Ranges 237
Sequences 237
Creating Sequences 238
Sequence operations 238
Efficient data processing 239
Hands-on: Put sequences to work 239
Iterators 240
Using for loops 241
Using forEach 242
Custom Iterators for complex data structures 242
Hands-on: use iterators 243
Ordering 244
Basic sorting 244
Natural ordering 245
Hands-on: sort products by price 246
Aggregate operations 247
Common aggregate functions 247
Custom aggregate operations 248
Considerations 249
Hands-on: Use aggregate operations 249
Transformations 251
map() 251
flatten() 252
flatMap() 252
associate() 253
Transformations to other data structures 253

Hands-on: Use transformations *253*
Filtering and searching 253
Filtering *254*
Searching *254*
Checking existence *254*
Hands-on: Use filter to find products *255*
Grouping 256
Basic grouping *256*
Grouping by multiple criteria *256*
Grouping operations *256*
Hands-on: Count products by category *257*
Handling missing elements 258
Combining and splitting collections 259
Conclusion 260

7. Scope Functions 261
Introduction 261
Structure 262
Objectives 262
The \`let\` scope function 263
Null safety *264*
Avoiding global scope pollution *264*
Transformtion *265*
Hands-on: use \`let\` scope function *266*
The \`with\` scope function 271
Usage in a nutshell *271*
Hands-on: use \`with\` scope function *273*
The \`run\` scope function 276
\`run\` for null safety 277
Hands-on: leveraging \`run\` for improved logic *277*
The \`apply\` scope function 278
Without \`apply\` *279*
With \`apply\` *280*
The \`also\` scope function 280
Performing side-effects on objects *281*

Executing temporary operations on objects ... 281
A cleaner syntax for chaining operations ... 281
Hands-on: use `also` scope function ... 282
Conclusion ... 285

8. Functional Programming ... 287
Introduction ... 287
Structure ... 288
Objectives ... 288
Introduction to functional programming ... 288
Higher-order functions and lambda functions ... 290
Higher-order functions ... 290
Lambda functions ... 291
Error handling ... 292
Testing ... 293
Hands-on: Using higher order and lambda functions ... 293
Setting up KotlinVoyager in IntelliJ ... 293
Implementing destinations model and repository ... 294
Implementing user interface ... 295
Functional interfaces ... 299
Functional interface ... 300
Functional interface with invoke operator ... 301
Hands-on: Leverage functional interface ... 301
Extension functions and properties ... 302
Extension functions ... 303
Extension properties ... 303
Common use cases ... 304
UI considerations ... 305
Error handling considerations ... 305
Hands-on: Creating utility extensions ... 305
Infix functions and operator overloading ... 306
Infix functions ... 306
Operator overloading ... 307
Hands-on: Infix functions and operator overloading ... 307
Builders and Kotlin DSL ... 311

Builder ... 311
DSL ... 312
Using Kotlin for DSLs ... 313
Building a DSL ... 314
Performance considerations ... 315
Effective testing of DSLs ... 316
Hands-on: Developing a DSL to create destinations ... 316
Enhancing object model ... 316
Implementing Destination DSL ... 317
Enhancing UI with manage destinations functionality ... 318
Using your DSL ... 320
Implementing Manage Destinations tab ... 321
Function composition ... 322
Mathematical foundation ... 323
Simple domino analogy ... 323
Example ... 323
Error handling and Function composition ... 324
Hands-on: Use function composition ... 325
Utilizing function composition ... 325
Conclusion ... 328

9. Exploring Delegation Design Pattern ... 329
Introduction ... 329
Structure ... 330
Objectives ... 330
Delegation pattern ... 330
Benefits of using delegation over inheritance ... 330
Comparing Delegation and other design patterns ... 331
Delegation pattern in Kotlin ... 331
Function delegation ... 332
Hands-on: Utilizing function delegation ... 334
Set up KotlinBrew in IntelliJ ... 334
Implement function delegation ... 335
Implementing user interface ... 336
Integrating UI components and running the application ... 340

Property delegation 340
Lazy property delegation 341
Observable property delegation 342
Property-to-property delegation 343
Map-based property delegation 344
Custom property delegation 345
Immutable properties 345
Mutable properties 346
Local property delegation 348
State management 350
Hands-on: Leveraging property delegate 351
Conclusion 353

10. Concurrency and Parallelism 355
Introduction 355
Structure 355
Objectives 356
Threads, processes, and beyond 356
Understanding through F1 analogies 356
Traditional threads 357
Virtual threads 357
Virtual threads versus Coroutines 358
Overview of coroutines 360
Suspending functions 360
Coroutine builders 361
Execution control 361
Starting a coroutine 361
Pausing and resuming 362
Cancellation 362
Timeouts 364
Coroutine scope, lifecycle, and structured concurrency 365
Coroutines context and dispatcher 367
Dispatchers and their roles 367
Confined versus unconfined dispatchers 369
Custom dispatchers 370

Context management 371
Elements of coroutine context 372
Manipulating Coroutine context 372
Hands-on: Adding coroutines to F1RaceSimulator 374
Preparing the application 375
Adding concurrency with coroutines 377
Adding realism with time delays in laps and pit stops 379
Synchronization 381
Mutex 381
Channels for data synchronisation 382
Thread confinement 384
Thread-safe data structures 385
Atomic data structures 385
AtomicInteger and AtomicLong 386
AtomicReference 387
Hands-on: Introduce safe management of shared data 387
Channels 388
Flows 391
Understanding flows 392
Hands-on: Using flow to track fastest lap 394
Flows versus Channels 396
Deciding between flows and channels 397
Choosing flows 397
Choosing channels 398
Exception handling 399
try-catch blocks 400
CoroutineExceptionHandler 400
Supervisor jobs 401
Supervision scope 401
Hands-on: Use exception handler and supervision 402
Conclusion 406

11. Unit Testing in Kotlin 407
Introduction 407
Structure 408

Objectives 408

Unit testing basics 408

- *Importance in software development* 408
- *Testing pyramid and test coverage* 409
- Unit testing in Test Driven Development 410

Testing frameworks overview 411

- *JUnit 5: The trustworthy standard* 411
- *kotlin-test: Tailored for Kotlin* 411
- *Kotest: The versatile framework* 412
- *Choosing the test framework* 412
- *Hands-on: setting up project for kotlin-test* 413

Basics of kotlin-test 413

- *Setting up the test environment* 413
- *Writing a simple test* 414
- *IDE Integration* 415
- *Hands-on: add tests to KotlinVoyager* 415
  - *Test Destination class* 415
  - *Test DestinationRepository class* 417

Mocking with MockK 418

- *Mockito vs. MockK for Kotlin* 419
- *Setting up mocks* 419
- *Stubbing and verifying* 420
  - *Given: stubbing the mock* 421
  - *When: acting on the System Under Test* 421
  - *Then: assertions and verifications* 421
- *Resetting mocks after tests* 423
- *Hands-on: using MockK in a test* 424

Advanced testing techniques 426

- *Mocking objects with MockK* 426
- *Mocking extension functions with MockK* 427
- *Mocking static methods with MockK* 428
- *Testing exception handling* 430
- *Asynchronous testing with coroutines* 432
- *Hands-on: Add unit test for F1RaceSimulator* 433
  - *Update gradle dependencies* 433

Add test for getRandomSponsor top level function 434
Add test for lap simulation 435
Test Suite effectiveness and management 436
Code coverage with JaCoCo 437
Hand-on: add JaCoCo coverage to F1RaceSimulator 438
Conclusion 440

12. Building a Simple REST API 441
Introduction 441
Structure 442
Objectives 442
Introduction to REST API with Ktor 442
REST APIs 442
Statelessness and resource orientation 442
CRUD operations 443
Mapping CRUD to HTTP methods 443
URI design for resource management 443
Common response codes 444
Using OpenAPI and Swagger in REST APIs 444
Choosing Ktor for the project 445
Spring Boot 446
Ktor 447
Advantages and considerations: Ktor vs. Spring Boot 448
Setting up the development environment 449
Generate ktor project 449
For IntelliJ community edition users 449
For IntelliJ Ultimate users 450
Inspect the project contents 452
Run hello world 454
Developing the KotlinBeats music API 456
Implementing GET: Retrieve all songs 457
Implementing POST: Add a new song 462
Step 1: Define the POST endpoint in OpenAPI 462
Step 2: Implement the endpoint 464
Step 3: Testing the POST endpoint 464
Step 4: Running your application 465

Implementing GET: Retrieve a single song by ID 466

Step 1: Define Endpoint in OpenAPI 466

Step 2: Implement the endpoint 467

Step 3: Test the endpoint 468

Step 4: Run and test in Swagger UI 468

Implementing PUT: Update an entire song 470

Step 1: Define the PUT endpoint in OpenAPI 471

Step 2: Implementing the PUT logic 471

Step 3: Verify the endpoint with a unit test 472

Step 4: Running and testing the PUT endpoint 473

Implementing PATCH: Partially update a song 474

Step 1: Defining PATCH endpoint in OpenAPI 474

Step 3: Testing the PATCH endpoint 476

Step 4: Verifying functionality in Swagger UI 477

Implementing DELETE - Remove a song 477

Step 1: Defining DELETE endpoint in OpenAPI 477

Step 2: Implementing DELETE logic in Routing.kt 477

Step 3: Testing DELETE endpoint 478

Step 4: Testing in Swagger UI 479

Conclusion 479

13. Building Event-Driven Cloud Native Serverless Application 481

Introduction 481

Structure 482

Objectives 482

Introduction to serverless computing 482

Serverless cloud 483

Serverless computing use cases 483

Challenges of serverless computing 484

Testing locally and the testing pyramid 484

Potential for complex architectures 484

Cold starts 485

Security and vendor lock-in 485

Overview of KotlinDocsFlow application 486

Problem statement 486

Automating submissions with KotlinDocsFlow 486

Insights into future development paths *488*
Choosing technologies 488
Choosing a cloud provider *489*
Infrastructure as code *490*
Choosing the framework 491
CI/CD pipeline *492*
Technology selection summary *493*
Solution design 493
Setting up the development environment 496
AWS Account *496*
User setup *496*
Creating an IAM User with AdministratorAccess permissions *497*
GitHub repository *498*
GitHub account *498*
Setting up GitHub repository *498*
Connecting GitHub to AWS *499*
Installing git *499*
Set up KotlinDocsFlow project *500*
Create the project *500*
Configuring the root build.gradle.kts *501*
Organizing project modules *502*
Integrating modules in Gradle *504*
Building the project *505*
Push your project to GitHub *505*
Developing the KotlinDocsFlow application 505
Creating function *506*
Creating Infrastructure as Code *507*
Configuring the AWS Resources *507*
Developing the CDK application *509*
CDK configuration *510*
Creating pipeline *511*
Deploying and testing in the cloud *515*
Adding email with link to the document *516*
Update your infrastructure code *517*
Update your function code *517*

Adding persistence and GraphQL API *520*
Update your function code *520*
Update infrastructure code *522*
Deploy and test *525*
Deleting resources and disabling user access *526*
Local execution, unit and integration testing *526*
Conclusion 527

Index 529-540

CHAPTER 1
Discovering the Power of Kotlin Programming

Introduction

This chapter provides an introduction to Kotlin, a modern multi-platform general purpose programming language. The chapter starts by explaining what Kotlin is, including its key features and benefits compared to other programming languages. It then dives into the advantages of using Kotlin, such as its concise syntax, null safety, interoperability with Java, and support for functional programming. The chapter also discusses the growing popularity of Kotlin in the industry and its potential for future development. Additionally, we will explain the **learn by doing** hands-on approach of this book. By the end of this chapter, readers will have a good understanding of Kotlin's advantages, its multi-platform capabilities, and why it is an attractive choice for modern software development, including frontend, backend and cloud computing.

Structure

This chapter covers the following topics:

- Kotlin
- Advantages of using Kotlin
- Learn by doing approach of this book

Objectives

This chapter introduces you to the world of Kotlin programming, where you will learn about an innovative and adaptable language that is changing the way developers build applications. JetBrains developed Kotlin, an open-source language that is rapidly growing in popularity in the tech industry. This chapter will walk you through Kotlin's key features and the practical benefits it has over other languages. You will learn why Kotlin is an excellent choice for a variety of projects, including backend development, cloud computing, mobile, desktop, web development, and data analytics.

Kotlin

Kotlin is an open-source, all-purpose, multi-platform, statically typed programming language created by JetBrains, suitable for a wide range of development needs. From building engaging mobile, desktop, and web applications to robust backend systems, including cloud-native solutions and data science projects, Kotlin's versatility is exceptional.

The language is designed to be fully compatible with Java, allowing it to take use of the vast ecosystem of Java libraries and frameworks, making it an excellent alternative for developers wishing to move to a more concise and expressive language while maintaining Java's functional richness.

Kotlin's ability to target multiple platforms derives from its ability to compile to bytecode for the **Java Virtual Machine** (**JVM**), translate to **JavaScript** for web development, and generate **native** code using LLVM or GraalVM for resource-constrained situations and quick startup times. This makes Kotlin an ideal choice for creating multi-platform mobile applications as well as server-side applications, particularly in cloud-native ecosystems where performance and resource management can be crucial.

Kotlin offers clarity and conciseness to data science, making it easy to manipulate data and perform difficult computations. Its static type system and powerful features, like as extension functions and lambdas, allow developers to construct clean and maintainable analytical applications.

The Kotlin Foundation **https://kotlinfoundation.org** controls the language's development to ensure its long-term sustainability, which was significantly enhanced when Google chose Kotlin as its preferred language for Android app development in May 2019. This support has resulted in a fast rise in Kotlin's popularity and acceptance for both mobile as well as backend development.

By adopting Kotlin, developers can benefit from a language that is not only modern and powerful but also supported by a large community and strong corporate backing, assuring a rewarding learning curve and an investment in skills that will be relevant for years to come.

Advantages of using Kotlin

In this section, we look at the numerous benefits of using Kotlin, a language that integrates the principles of object-oriented and functional programming with safety features and multiplatform flexibility.

All-around language capabilities

Kotlin is a modern language that supports **object-oriented** (**OO**) and **functional programming (FP)** from the ground up. Kotlin is designed to avoid the drawbacks of languages like Java, which introduced functional programming features long after its origin, and JavaScript, which retrofitted object-oriented capabilities, by providing a more accessible approach to both paradigms. Kotlin integrates both programming paradigms, making software development not only more pleasant but also more robust, testable, and safer. This dual capacity means that Kotlin developers always have the correct tool for the job, making the language extremely adaptable and enjoyable once mastered.

Object-oriented programming in Kotlin

Object-oriented programming (**OOP**) is a paradigm that uses **objects** to store data (**properties**) and code (**methods**). OO programming leverages:

- **Encapsulation**: ensuring that data structures and operations are used as intended.
- **Inheritance**: the process of directly obtaining the attributes and behavior of other objects in order to create a new object.
- **Polymorphism**: the ability of methods to behave differently depending on the object on which they operate, enabling a single interface to be used for different underlying data types.

Kotlin also integrates built-in design patterns like singleton and delegation directly into its type system, making typical OO patterns easier to implement and less error-prone.

In this book, you can learn about object-oriented programming with Kotlin in *Chapter 5, Object-Oriented Programming* and explore Kotlin's delegation pattern in *Chapter 9, Exploring Delegation Design Pattern*.

Functional programming in Kotlin

In Kotlin, functional programming focuses on developing software with pure functions, which have no side effects and do not rely on or change any state outside their scope. This programming approach has many benefits:

- **Immutability**: reducing runtime errors by not changing the state once it has been created. This strategy reduces runtime errors, making the code more predictable and debuggable.
- **Higher-order functions**: increasing the language's versatility by letting functions to be provided as arguments, or returned as values. This feature improves the language's flexibility and power, allowing for more modular and expressive programming.
- **Expressiveness**: allowing developers to produce more concise and readable code. This minimizes the number of defects and the effort required to read and maintain the code.

Kotlin also introduces scope functions that bring additional clarity and brevity to operations that would be more verbose in other languages. This book covers scope functions in *Chapter 7, Scope Functions*, and functional programming in *Chapter 8, Functional Programming*.

Safety features

Kotlin has numerous safety features which inherently make the code less buggy and easier to read and understand:

- **Null safety**: Kotlin's type system contains nullability checks, which the compiler enforces to avoid **`NullPointerExceptions`**. This strict handling of null values means that all null-related operations are explicitly handled, improving the safety and stability of your applications.
- **Safe casting**: Kotlin includes safe casting capabilities to avoid **`ClassCastException`**. This feature guarantees that objects are cast appropriately, increasing application stability by preventing frequent type mistakes.
- **Immutability**: Kotlin advocates the usage of immutable objects, which improves safety by preventing objects from being modified after they are created. This immutability results in more predictable code, which is easier to debug and maintain.
- **Structured concurrency**: Kotlin's concurrency model is structured, making it easier to manage multiple operations that execute concurrently. This methodology encourages simpler and more efficient concurrent code, eliminating the complexity typically associated with asynchronous programming.
- **Smart casts and type safety**: Kotlin's type system is inherently safe since variables must be explicitly declared and handled within their type constraints. On top of this, Kotlin provides smart casts, which decrease boilerplate and improve code readability by automatically casting types once their conditions have been verified. This enables you to deal with variables more fluidly without sacrificing safety, ensuring that operations are both safe and straightforward. For example, after a

variable is confirmed as non-null or of a certain type, Kotlin smartly casts it within the condition block, simplifying subsequent actions and lowering the margin for errors.

The book discusses safety features in *Chapter 4, Fundamental Building Blocks of Kotlin,* and structured concurrency in *Chapter 10, Concurrency and Parallelism*.

Multiplatform development

Kotlin's multiplatform capabilities let developers create applications that operate on variety of platforms while reusing common code, resulting in reduced development time and effort. This functionality is very useful for developing frontend and backend applications that need to work across many platforms.

Frontend development: Kotlin allows you to build sophisticated, responsive front-end applications for Android, iOS, desktop, and the web. Jetpack Compose Multiplatform is one of the standout technologies, allowing you to create declarative UIs for Android, iOS, desktop and the web from a single codebase. For further information, see Jetpack Compose Multiplatform **https://www.jetbrains.com/lp/compose-multiplatform** and Kotlin Wasm **https://kotlinlang.org/docs/wasm-overview.html**.

Backend development: Kotlin's adaptability extends to the backend, where it can compile to the JVM, allowing for easy integration with Java and its ecosystem. For situations that need specific languages, such as certain language restrictive cloud-native components, Kotlin can be compiled to JavaScript using NodeJS to ensure broad compatibility, see **https://kotlinlang.org/docs/js-project-setup.html**. Furthermore, Kotlin may be built to native binaries using LLVM-based backend **https://kotlinlang.org/docs/native-overview.html** or GraalVM **https://www.graalvm.org** in circumstances demanding optimized memory consumption and faster startup times.

You will use Jetpack Compose in hands-on projects of *Chapter 8, Functional Programming,* and *Chapter 9, Exploring Delegation Design Pattern*, create a REST API backend application in *Chapter 12, Building a Simple REST API*, and develop, deploy and run a Kotlin application on cloud native in *Chapter 13, Building Event Driven Cloud Native Serverless Application*.

Interoperability with Java

One of Kotlin's most important characteristics is its compatibility with Java, which allows it to smoothly integrate into existing Java environments. This compatibility has various practical benefits:

- **Seamless integration on JVM**: Kotlin's seamless integration with the **Java Virtual Machine** (**JVM**) allows it to run anywhere Java can, including any cloud platforms that support the Java language. This makes Kotlin an ideal choice for enterprises and developers that want to use the current infrastructure without making major modifications.