

# Index

## A

- Activity component, [The Activity and Its Friends-The back stack](#)
  - back stack, [The back stack](#)
  - fragments, [Fragments](#)
- Activity context, [Component context](#)
- Actor model, CSP, [Similarities with the Actor Model](#)
- Alarms, [Energy Profiler](#)
- Analysis Panel, [Analysis panel-Analysis panel](#)
- Android applications
  - Activity component, [The Activity and Its Friends-The back stack](#)
  - back stack, [The back stack](#)
  - basics, [Applications](#)
  - bound services, [Bound Services-Bound Services](#)
  - broadcast receivers, [Broadcast Receivers](#)
  - components of, [Android Application Components: The Building Blocks-Broadcast Receivers](#)
  - content providers, [Content Providers](#)
  - environment, [The Android Application Environment-Application context](#)
  - fragments, [Fragments](#)
  - services, [Services-Bound Services](#)
  - started services, [Started Services](#)
- Android fundamentals, [Android Fundamentals-Summary](#)
  - Activity component, [The Activity and Its Friends-The back stack](#)
  - Android stack, [The Android Stack-Applications](#)
  - application architectures, [Android Application Architectures-The Local Model](#)
  - application components, [Android Application Components: The Building Blocks-Broadcast Receivers](#)
  - application context, [Application context](#)
  - application environment, [The Android Application Environment-Application context](#)

- applications, [Applications](#)
- broadcast receivers, [Broadcast Receivers](#)
- component context, [Component context](#)
- content providers, [Content Providers](#)
- context, [Context-Application context](#)
- hardware, [Hardware](#)
- intents and intent filters, [Intents and Intent Filters](#)
- kernel, [Kernel](#)
- Local Model, [The Local Model](#)
- Model-View-Intent, [Model-View-Intent](#)
- Model-View-Presenter, [Model-View-Presenter](#)
- Model-View-View Model, [Model-View-ViewModel](#)
- MVC, [MVC: The Foundation](#)
- patterns, [Android Patterns-Model-View-ViewModel](#)
- runtime environment, [Android Runtime Environment](#)
- services, [Services-Bound Services](#)
- system services, [System Services](#)
- widgets, [Widgets](#)
- Android Profiler, [Android Profiler-Memory Profiler](#)
  - about, [Android Profiler-Android Profiler](#)
  - Analysis Panel, [Analysis panel-Analysis panel](#)
  - CPU Profiler, [CPU Profiler-Recording a sample method trace](#)
  - CPU timeline, [CPU timeline](#)
  - Energy Profiler, [Energy Profiler-Energy Profiler](#)
  - Memory Profiler, [Memory Profiler-Memory Profiler](#)
  - method tracing, [Method tracing-Method tracing](#)
  - network calls and, [Network call, expanded: Overview | Response | Request | Callstack](#)-[Network call, expanded: Overview | Response | Request | Callstack](#)
  - Network Profiler, [Network Profiler-Network call, expanded: Overview | Response | Request | Callstack](#)
  - recording a Sample Method Trace, [Recording a sample method trace-Recording a sample method trace](#)
  - thread activity timeline, [Thread activity timeline](#)
  - viewing network calls with Connection View and Thread View, [Viewing network calls with Connection View and Thread View](#)

- Android profiling tools, [Performance Considerations with Android Profiling Tools-Summary](#)
  - Android Profiler, [Android Profiler-Memory Profiler](#)
  - CPU Profiler, [CPU Profiler-Recording a sample method trace](#)
  - detecting memory leaks with LeakCanary, [Detecting Memory Leaks with LeakCanary-Detecting Memory Leaks with LeakCanary](#)
  - Energy Profiler, [Energy Profiler-Energy Profiler](#)
  - Memory Profiler, [Memory Profiler-Memory Profiler](#)
  - Network Profiler, [Network Profiler-Network call, expanded: Overview | Response | Request | Callstack](#)
- Android stack, [The Android Stack-Applications](#)
  - applications, [Applications](#)
  - hardware, [Hardware](#)
  - kernel, [Kernel](#)
  - runtime environment, [Android Runtime Environment](#)
  - system services, [System Services](#)
- any() function, [The Boolean Functions](#)
- application environment, [The Android Application Environment-Application context](#)
  - application context, [Application context](#)
  - component context, [Component context](#)
  - context, [Context-Application context](#)
  - intents/intent filters, [Intents and Intent Filters](#)
- ApplicationContext, [Application context](#)
- ArrayList, [Traditional Approach Using java.util.concurrent.ExecutorService](#)
- ART, [System Services](#)
- asSharedFlow(), [Create a SharedFlow](#)
- async, [Using Coroutines and Suspending Functions: A Practical Example](#)
- async coroutine builder, [The async Coroutine Builder-The async Coroutine Builder](#)
- atomicity, [Atomicity](#)
- automatic cancellation, [Automatic Cancellation](#)
- await(), [Cancelling a Task Delegated to a Third-Party Library](#)

- back pressure, [Back Pressure-Back Pressure](#), [Back Pressure](#)
- back stack, [The back stack](#)
- Binder, [System Services](#)
- Bitmap pooling, [Bitmap Pooling and Caching](#)
- blocking call, [Blocking Call Versus Nonblocking Call](#)
- BlockingQueue, [Channels Overview](#)
- Boolean functions, [The Boolean Functions](#)
- bound services, [Bound Services-Bound Services](#)
- broadcast receivers, [Broadcast Receivers](#)
- buffered channels, [Buffered Channel](#)
- BufferOverflow, [Suspend or not?](#)

## C

- caching, [Bitmap Pooling and Caching](#)
- Call Chart, [Thread activity timeline](#), [Method tracing](#)
- Call.enqueue(), [Cancelling a Task Delegated to a Third-Party Library](#)
- callback-based API, [Use Case #1: Interface with a Callback-Based API-Use Case #1: Interface with a Callback-Based API](#)
- callbackFlow builder, [Use Case #1: Interface with a Callback-Based API](#)
- callbacks, [Handling Concurrency Using Callbacks-Summary](#)
  - app creation, [Creating the App-Memory leaks](#)
  - example-of-purchase feature, [Example-of-Purchase Feature-Example-of-Purchase Feature](#)
  - logic implementation, [Implement the Logic](#)
  - memory leaks and, [Memory leaks](#)
  - structured concurrency and, [Structured concurrency](#)
  - threading model limitations, [Limitations of the Threading Model](#)
  - view, [View-View](#)
  - ViewModel, [View-Model](#)
- CancellableContinuation, [Cancelling a Task Delegated to a Third-Party Library](#)
- cancellation, [Cancellation-Causes of Cancellation](#)
  - automatic, [Automatic Cancellation](#)
  - coroutine, [Cancelling a Coroutine-Cancelling a Coroutine](#)
  - coroutine lifecycle, [Coroutine Lifecycle-Job holds the state](#)
  - delay, [delay Is Cancellable](#)

- failure, [Cancellation](#)
- handling, [Handling Cancellation](#)
- task delegated to a third-party library, [Cancelling a Coroutine-Cancelling a Coroutine](#)
- CancellationException, [Coroutines That Are Cooperative with Cancellation](#)
- catch operator, [The catch Operator-You can use emit from inside catch](#)
  - declarative style, [The catch Operator](#)
  - emit a particular value, [You can use emit from inside catch](#)
  - example, [Another example](#)
  - exception transparency, [Exception transparency](#)
- channel.receive(), [Iterating over a Channel](#)
- channels, [Channels-Summary](#)
  - about, [Channels Overview-Channels Overview](#)
  - buffered, [Buffered Channel](#)
  - communicating sequential processes, [Communicating Sequential Processes-Final Thoughts](#)
  - conflated, [Conflated Channel](#)
  - fan-out and fan-in, [Fan-Out and Fan-In](#)
  - hot, [Channels Are Hot](#)
  - limitations of, [Limitations of Channels-Limitations of Channels](#)
  - performance test, [Performance Test-Performance Test](#)
  - produce builder with, [Channel Producers](#)
  - rendezvous, [Rendezvous Channel-Other flavors of Channel](#)
  - select expression, [The select Expression-The select Expression](#)
  - unlimited, [Unlimited Channel-Unlimited Channel](#)
- ChannelT(Channel.CONFLATED), [Conflated Channel](#)
- children CPU time, [Analysis panel](#)
- CircleView, [Reducing Unnecessary Work](#)
- class initialization, [Class Initialization-Class Initialization](#)
- classes, [Classes-Sealed Classes](#)
  - class initialization, [Class Initialization-Class Initialization](#)
  - companion objects, [Companion Objects](#)
  - data classes, [Data Classes](#)
  - delegates, [Delegates](#)
  - enum classes, [Enum Classes-Enum Classes](#)
  - lateinit properties, [lateinit Properties-lateinit Properties](#)

- lazy properties, [Lazy Properties](#)
- properties, [Properties](#)
- sealed classes, [Sealed Classes](#)
- CloseableCoroutineScope(..), [Using Suspending Functions and Coroutines](#)
- cold flows, [Examples of Cold Flow Usage-Usage](#)
  - concurrently transforming a stream of values, [Use Case #2: Concurrently Transform a Stream of Values-Final Thoughts](#)
  - creating a custom operator, [Use Case #3: Create a Custom Operator-Usage](#)
  - interfacing with a callback-based API, [Use Case #1: Interface with a Callback-Based API-Use Case #1: Interface with a Callback-Based API](#)
- collect, [A More Realistic Example](#)
- collections framework (Kotlin), [The Kotlin Collections Framework-Summary](#)
  - Boolean functions, [The Boolean Functions](#)
  - creating containers, [Creating Containers](#)
  - example, [Functional Versus Procedural: A Simple Example, An Example-The Implementation](#)
  - filter functions, [Filter Functions-Map](#)
  - flatMap, [flatMap](#)
  - functional Android, [Functional Android](#)
  - grouping, [Grouping](#)
  - iterators versus sequences, [Iterators Versus Sequences](#)
  - java interoperability, [Java Interoperability](#)
  - overloaded operators, [Overloaded Operators](#)
  - transformation functions, [Kotlin Transformation Functions-Iterators Versus Sequences](#)
- communicating sequential processes (CSP)
  - about, [Channels](#)
  - back pressure, [Back Pressure](#)
  - deadlock, [Deadlock in CSP-TL;DR](#)
  - example, [Communicating Sequential Processes-Final Thoughts](#)
  - fan-out and fan-in, [Fan-Out and Fan-In](#)
  - final thoughts, [Final Thoughts](#)

- first implementation, [A First Implementation-A First Implementation](#)
- model and architecture, [Model and Architecture](#)
- performance test, [Performance Test-Performance Test](#)
- putting it all together, [Putting It All Together](#)
- select expression, [The select Expression-The select Expression](#)
- sequential execution inside a process, [Execution Is Sequential Inside a Process](#)
- similarities with the actor model, [Similarities with the Actor Model](#)
- companion objects, [Companion Objects](#)
- concurrency, limiting, [A First Implementation](#)
- concurrent programming, [Concurrency in Android-Summary](#)
  - atomicity, [Atomicity](#)
  - dropped frames, [Dropped Frames-Dropped Frames](#)
  - Executors and ExecutorServices, [Executors and ExecutorServices-Executors and ExecutorServices](#)
  - job managing tools, [Tools for Managing Jobs-WorkManager](#)
  - JobScheduler, [JobScheduler-JobScheduler](#)
  - Looper/Handler, [Looper/Handler-Looper/Handler](#)
  - memory leaks, [Memory Leaks-Memory Leaks](#)
  - thread managing tools, [Tools for Managing Threads-Executors and ExecutorServices](#)
  - thread safety, [Thread Safety-Visibility](#)
  - threading model, [The Android Threading Model](#)
  - visibility, [Visibility](#)
  - WorkManager, [WorkManager](#)
- conflated channels, [Conflated Channel](#)
- Connection View, [Viewing network calls with Connection View and Thread View](#)
- ConstraintLayout, [Achieving Flutter View Hierarchy with ConstraintLayout](#)
  - achieving flutter view hierarchy, [Achieving Flutter View Hierarchy with ConstraintLayout-Achieving Flutter View Hierarchy with ConstraintLayout](#)
- containers, creating, [Creating Containers](#)
- content providers, [Content Providers](#)
- context, [Context-Application context](#)



- application, [Application context](#)
- component, [Component context](#)
- context preservation, [Use Case #1: Interface with a Callback-Based API](#)
- Context.getApplicationContext() method, [Application context](#)
- Continuation Passing Style (CPS), [Suspending Functions Under the Hood](#)
- Controller, [MVC: The Foundation](#)
- coroutine.start(), [Channels Are Hot](#)
- CoroutineContext (see context)
- CoroutineExceptionHandler, [Causes of Cancellation](#)
- coroutines, [Coroutines Concepts-Summary](#)
  - async coroutine builder, [The async Coroutine Builder-The async Coroutine Builder](#)
  - cancellation, [Coroutine cancellation](#), [Cancelling a Coroutine-Cancelling a Coroutine](#)
  - cooperative with cancellation, [Coroutines That Are Cooperative with Cancellation-Coroutines That Are Cooperative with Cancellation](#)
  - CoroutineScope and CoroutineContext, [CoroutineScope and CoroutineContext-CoroutineScope and CoroutineContext](#)
  - example, [Your First Coroutine-Your First Coroutine, Using Coroutines and Suspending Functions: A Practical Example-Using Coroutines and Suspending Functions: A Practical Example](#)
  - Job, [Job holds the state](#)
  - lifecycle, [Coroutine Lifecycle-Job holds the state](#)
  - structured concurrency with (see structured concurrency with coroutines)
  - suspending functions, [Suspending Functions-Suspending Functions Under the Hood](#), [Using Suspending Functions and Coroutines-Using Suspending Functions and Coroutines](#)
- CoroutineScope, [CoroutineScope and CoroutineContext, Using Suspending Functions and Coroutines](#)
- coroutineScope, [supervisorScope Builder, Use Case #3: Create a Custom Operator](#)
- CPS (Continuation Passing Style), [Suspending Functions Under the Hood](#)
- CPU Profiler, [CPU Profiler-Recording a sample method trace](#)



- Analysis Panel, [Analysis panel-Analysis panel](#)
- CPU timeline, [CPU timeline](#)
- method tracing, [Method tracing-Method tracing](#)
- recording a Sample Method Trace, [Recording a sample method trace-Recording a sample method trace](#)
- thread activity timeline, [Thread activity timeline](#)
- CSP (see communicating sequential processes)

## D

- daemon, [Broadcast Receivers](#)
- data classes, [Data Classes](#)
- deadlock, [The Android Threading Model](#), [Deadlock in CSP-TL;DR](#)
- declarative style, catch operator, [The catch Operator](#)
- delay function, [Your First Coroutine](#), [delay Is Cancellable](#), [Back Pressure](#)
- delegates, [Delegates](#)
- DEX, [The Android Application Environment](#)
- Dispatcher, [Using Suspending Functions and Coroutines](#)
- Dispatchers.Default, [CoroutineScope and CoroutineContext](#), [Using Coroutines and Suspending Functions: A Practical Example](#)
- Dispatchers.IO, [CoroutineScope and CoroutineContext](#)
- Dispatchers.Main, [CoroutineScope and CoroutineContext](#)
- display buffers, [Dropped Frames](#)
- downstream flow, [Use Case #1: Interface with a Callback-Based API](#)
- Drawables, [Reducing Programmatic Draws with Drawables-Reducing Programmatic Draws with Drawables](#)
- DrawableStates, [Reducing Programmatic Draws with Drawables](#)
- dropped frames, [Dropped Frames-Dropped Frames](#)

## E

- emit, [Send Values to the SharedFlow](#), [Suspend or not?](#)
- Energy Profiler, [Energy Profiler-Energy Profiler](#)
- ensureActive, [Cancelling a Coroutine](#), [delay Is Cancellable](#)
- enum classes, [Enum Classes-Enum Classes](#)
- equals, [Data Classes](#)

- error handling, [Error Handling-Exception Transparency Violation](#)
  - exception transparency violation, [Exception Transparency Violation](#)
  - separation of concern, [Separation of Concern Is Important](#)
  - try/catch block, [The try/catch Block-The try/catch Block](#)
- exception handling, [Exception Handling-Unhandled Exceptions](#)
  - exposed exceptions, [Exposed Exceptions-Exposed Exceptions](#)
  - materializing exceptions, [Materialize Your Exceptions-A bonus](#)
  - unhandled exceptions, [Unhandled Exceptions-Unhandled Exceptions](#)
  - unhandled versus exposed exceptions, [Unhandled Versus Exposed Exceptions-Unhandled Versus Exposed Exceptions](#)
- exception transparency, [Exception transparency](#)
- exception transparency violation, [Exception Transparency Violation](#)
- Executors, [Executors and ExecutorServices-Executors and ExecutorServices](#)
- Executors.newSingleThreadExecutor(), [View](#)
- ExecutorServices, [Executors and ExecutorServices-Executors and ExecutorServices](#)
- explicit intent, [Intents and Intent Filters](#)
- exposed exceptions, [Unhandled Versus Exposed Exceptions-Exposed Exceptions](#)
- extension functions, [Extension Functions-Extension Functions](#)
- extension properties, [Extension Functions](#)
- extraBufferCapacity, [Buffer values](#)

## F

- failure cancellation, [Cancellation](#)
- fan-in, [Fan-Out and Fan-In](#)
- fan-out, [Fan-Out and Fan-In](#)
- fetchImage, [Suspending Functions Under the Hood](#), [Materialize Your Exceptions](#)
- fetchProfile, [Don't Be Mistaken About the suspend Modifier](#)
- filesystem caches, [Bitmap Pooling and Caching](#)
- filter functions, [Filter Functions-Map](#)
- filterNot function, [Filter Functions](#)

- finish() method, [The Activity and Its Friends](#)
- Flame Chart, [Analysis panel](#)
- flatMap, [flatMap](#), [The Implementation](#)
- flatter view hierarchy, [Achieving Flatter View Hierarchy with ConstraintLayout](#)-[Achieving Flatter View Hierarchy with ConstraintLayout](#)
- flowOn, [Use Case #1: Interface with a Callback-Based API](#)
- @FlowPreview, [Use Case #2: Concurrently Transform a Stream of Values](#)
- flows, [Flows-Summary](#)
  - catch operator, [The catch Operator-You can use emit from inside catch](#)
  - cold flows, [Examples of Cold Flow Usage-Usage](#)
  - error handling, [Error Handling-Exception Transparency Violation](#)
  - example, [A More Realistic Example-A More Realistic Example](#)
  - exception transparency violation, [Exception Transparency Violation](#)
  - hot flows with SharedFlow, [Hot Flows with SharedFlow-An Example of StateFlow Usage](#)
  - materializing exceptions, [Materialize Your Exceptions-A bonus](#)
  - operators, [Operators](#)
  - separation of concern, [Separation of Concern Is Important](#)
  - terminal operators, [Terminal Operators](#)
  - try/catch block, [The try/catch Block-The try/catch Block](#)
  - use case: concurrently transforming a stream of values, [Use Case #2: Concurrently Transform a Stream of Values-Final Thoughts](#)
  - use case: creating a custom operator, [Use Case #3: Create a Custom Operator-Usage](#)
  - use case: interfacing with a callback-based API, [Use Case #1: Interface with a Callback-Based API-Use Case #1: Interface with a Callback-Based API](#)
- forEach method, [Functional Versus Procedural: A Simple Example](#)
- ForkJoinPool, [Executors and ExecutorServices](#)
- fragments, [Fragments](#)
- function types, [Function Types-Function Types](#)
- functional Android, [Functional Android](#)
- functional programming

- Android and, [Functional Android](#)
- procedural versus, [Functional Versus Procedural: A Simple Example](#)
- Future.get(), [The async Coroutine Builder](#)

## G

- garbage collection (GC), [Memory Profiler](#)
- generators, [Iterators Versus Sequences](#)
- generics, [Generics](#)
- getDataFlow, [A More Realistic Example](#)
- GoogleLocationProvider, [Energy Profiler](#)
- GPX Record, [Memory Profiler](#)
- GridLayout, [Achieving Flutter View Hierarchy with ConstraintLayout](#)
- groupBy function, [Grouping](#)

## H

- HandlerThread, [A Reminder About HandlerThread-A Reminder About HandlerThread](#)
- hardware, [Hardware](#)
- hashCode, [Data Classes](#)
- heap dumps, [Android Profiler](#), [Memory Profiler](#)
- higher-order functions, [Function Types](#)
- Hilt, [Reducing Unnecessary Work](#)
- hot channels, [Channels Are Hot](#)
- hot flows, [Hot Flows with SharedFlow-An Example of StateFlow Usage](#)
  - create a SharedFlow, [Create a SharedFlow](#)
  - register a subscriber, [Register a Subscriber](#)
  - send values to the SharedFlow, [Send Values to the SharedFlow](#)
  - SharedFlow as an event bus, [Using SharedFlow as an Event Bus](#)
  - SharedFlow to stream data, [Using SharedFlow to Stream Data-Buffer values](#)
  - StateFlow, [StateFlow: A Specialized SharedFlow-An Example of StateFlow Usage](#)

## I

- identifier field, [Properties](#)
- in-memory caches, [Bitmap Pooling and Caching](#)
- IntentFilter, [Intents and Intent Filters](#)
- intents/intent filters, [Intents and Intent Filters](#)
- interface with a callback-based API, [Use Case #1: Interface with a Callback-Based API-Use Case #1: Interface with a Callback-Based API](#)
- invariants, [Invariants-Thread-Safe Collections](#)
  - mutexes, [Mutexes](#)
  - thread-safe collections, [Thread-Safe Collections-Thread-Safe Collections](#)
- isActive, [Cancelling a Coroutine](#)
- iterating over a channel, [Iterating over a Channel-Iterating over a Channel](#)
- iterators, sequences, [Iterators Versus Sequences](#)

## J

- java interoperability, [Java Interoperability](#)
- Java Native Interface (JNI), [Android Runtime Environment](#)
- job managing tools, [Tools for Managing Jobs-WorkManager](#)
  - JobScheduler, [JobScheduler-JobScheduler](#)
  - WorkManager, [WorkManager](#)
- Job.cancel, [Causes of Cancellation](#)
- job.cancelAndJoin(), [Cancelling a Task Delegated to a Third-Party Library](#)
- job.join(), [Conflated Channel](#)
- job.start(), [Coroutine Lifecycle](#)
- JobInfo, [JobScheduler](#)
- Jobs, [Energy Profiler](#)
- JobScheduler, [JobScheduler-JobScheduler](#)
- joinToString function, [The Implementation](#)
- JSON, [Minimizing Asset Payload in Network Calls](#)

## K

- kernel, [Kernel](#)
- Kotlin (basics), [Kotlin Essentials-Summary](#)

- class initialization, [Class Initialization-Class Initialization](#)
- classes, [Classes-Sealed Classes](#)
- companion objects, [Companion Objects](#)
- data classes, [Data Classes](#)
- delegates, [Delegates](#)
- enum classes, [Enum Classes-Enum Classes](#)
- extension functions, [Extension Functions-Extension Functions](#)
- function types, [Function Types-Function Types](#)
- generics, [Generics](#)
- lambdas, [Lambdas](#)
- lateinit properties, [lateinit Properties-lateinit Properties](#)
- lazy properties, [Lazy Properties](#)
- null safety, [Null Safety-Null Safety](#)
- primitive types, [Primitive Types](#)
- properties, [Properties](#)
- sealed classes, [Sealed Classes](#)
- unit type, [The Unit Type](#)
- variables, [Variables](#)
- variables and functions, [Variables and Functions-Extension Functions](#)
- visibility modifiers, [Visibility Modifiers-Visibility Modifiers](#)
- kotlin.collections.immutable library, [Mutability](#)
- kotlin.coroutines library, [CoroutineScope and CoroutineContext](#)

## L

- lambdas, [Lambdas](#)
- lateinit properties, [lateinit Properties-lateinit Properties](#)
- launch {..} function, [Coroutine Lifecycle](#)
- Layout Inspector, [Achieving Flatter View Hierarchy with ConstraintLayout](#)
- LayoutInflater, [Reducing Programmatic Draws with Drawables](#)
- lazy initialization, [Lazy Properties](#)
- lazy properties, [Lazy Properties](#)
- LeakCanary, [Detecting Memory Leaks with LeakCanary-Detecting Memory Leaks with LeakCanary](#)

- LinearLayout, [Achieving Flatter View Hierarchy with ConstraintLayout](#)
- LiveData, [Model–View–ViewModel](#), [Memory Leaks](#), [The implementation](#)
- Local Model, [The Local Model](#)
- logic implementation, [Implement the Logic](#)
- Looper class, [A Reminder About HandlerThread](#)
- Looper.prepare(), [Looper/Handler](#)
- Looper.start(), [Looper/Handler](#)
- Looper/Handler, [Looper/Handler-Looper/Handler](#)

## M

- map function, [Map-Map](#)
- mapIndexed.mapIndexed, [Map](#)
- mapOf function, [Creating Containers](#)
- materializing exceptions, [Materialize Your Exceptions-A bonus](#)
- memory leaks, [Memory Leaks-Memory Leaks](#)
  - in app creation, [Memory leaks](#)
  - LeakCanary detecting, [Detecting Memory Leaks with LeakCanary-](#)  
[Detecting Memory Leaks with LeakCanary](#)
- Memory Profiler, [Memory Profiler-Memory Profiler](#)
- method tracing, [Android Profiler](#), [Method tracing-Method tracing](#)
- minification, with R8 and ProGuard, [Minification and Obfuscation with R8 and ProGuard](#)
- Model, defined, [MVC: The Foundation](#)
- Model–View–Controller (MVC) pattern, [MVC: The Foundation](#)
- Model–View–Intent (MVI) pattern, [Model–View–Intent](#)
- Model–View–Presenter (MVP) pattern, [Model–View–Presenter](#)
- Model–View–View Model (MVVM) pattern, [Model–View–ViewModel](#)
- MutableSharedFlow(), [Create a SharedFlow](#)
- mutexes, [Mutexes](#)
- MVC (Model–View–Controller) pattern, [MVC: The Foundation](#)
- MVI (Model–View–Intent) pattern, [Model–View–Intent](#)
- MVP (Model–View–Presenter) pattern, [Model–View–Presenter](#)
- MVVM (Model–View–View Model) pattern, [Model–View–ViewModel](#)



## N

- network calls
  - about, [Network call, expanded: Overview | Response | Request | Callstack](#)
  - minimizing asset payload in, [Minimizing Asset Payload in Network Calls](#)
  - viewing, [Viewing network calls with Connection View and Thread View](#)
- Network Profiler, [Network Profiler-Network call, expanded: Overview | Response | Request | Callstack](#)
  - and network calls, [Network call, expanded: Overview | Response | Request | Callstack](#)
  - viewing network calls with Connection View and Thread View, [Viewing network calls with Connection View and Thread View](#)
- NewsDao, [The implementation](#)
- nonblocking call, [Blocking Call Versus Nonblocking Call](#)
- null safety, [Null Safety-Null Safety](#)

## O

- obfuscation, with R8 and ProGuard, [Minification and Obfuscation with R8 and ProGuard](#)
- object-oriented programming (OOP), [Functional Programming](#)
- offer, [Back Pressure](#)
- OkHttp, [Bitmap Pooling and Caching](#)
- onCreate, [lateinit Properties](#), [The Activity and Its Friends](#), [Services](#), [Summary](#)
- onCreateView, [lateinit Properties](#)
- onDestroy, [The Activity and Its Friends](#), [Services](#)
- onDraw, [Reducing Unnecessary Work](#)
- onResume, [Summary](#)
- onStartJob, [JobScheduler](#)
- OOP (object-oriented programming), [Functional Programming](#)
- operators, [Operators](#)

- create custom, [Use Case #3: Create a Custom Operator-Usage](#)
- terminal, [Terminal Operators](#)
- overloaded operators, [Overloaded Operators](#)

## P

- parallel decomposition, [Parallel Decomposition](#)
- parent context, [CoroutineScope and CoroutineContext](#)
- patterns, Android, [Android Patterns-Model-View-ViewModel](#)
  - Model-View-Intent, [Model-View-Intent](#)
  - Model-View-Presenter, [Model-View-Presenter](#)
  - Model-View-View Model, [Model-View-ViewModel](#)
- performance optimizations, [Trimming Down Resource Consumption with Performance Optimizations-Summary](#)
  - achieving flatter view hierarchy with ConstraintLayout, [Achieving Flatter View Hierarchy with ConstraintLayout-Achieving Flatter View Hierarchy with ConstraintLayout](#)
  - Bitmap pooling and caching, [Bitmap Pooling and Caching](#)
  - minification and obfuscation with R8 and ProGuard, [Minification and Obfuscation with R8 and ProGuard](#)
  - minimizing asset payload in network calls, [Minimizing Asset Payload in Network Calls](#)
  - reducing programmatic draws with Drawables, [Reducing Programmatic Draws with Drawables-Reducing Programmatic Draws with Drawables](#)
  - reducing unnecessary work, [Reducing Unnecessary Work-Reducing Unnecessary Work](#)
  - using static functions, [Using Static Functions](#)
- performance test, [Performance Test-Performance Test](#)
- POJOs (plain old Java objects), [Data Classes](#)
- primary constructor, [Class Initialization](#)
- primitive types, [Primitive Types](#)
- private visibility modifiers
  - in Java, [Visibility Modifiers](#)
  - in Kotlin, [Visibility Modifiers](#)
- procedural programming, [Functional Versus Procedural: A Simple Example](#)

- produceValues(), [Channel Producers](#)
- profileDeferred.await, [Using Coroutines and Suspending Functions: A Practical Example](#)
- programmatic draws, [Reducing Programmatic Draws with Drawables-Reducing Programmatic Draws with Drawables](#)
- ProGuard, [Minification and Obfuscation with R8 and ProGuard](#)
- properties, [Properties](#)
- protected visibility modifiers
  - in Java, [Visibility Modifiers](#)
  - in Kotlin, [Visibility Modifiers](#)
- public visibility modifiers
  - in Java, [Visibility Modifiers](#)
  - in Kotlin, [Visibility Modifiers](#)
- PurchaseViewModel, [Example-of-Purchase Feature](#)

## Q

- query(), [Content Providers](#)
- quitSafely, [A Reminder About HandlerThread](#)

## R

- R8, [Minification and Obfuscation with R8 and ProGuard](#)
- RAM, [Memory Profiler](#)
- RelativeLayout, [Achieving Flatter View Hierarchy with ConstraintLayout](#)
- RenderThread, [Method tracing](#)
- rendezvous channels, [Rendezvous Channel-Other flavors of Channel](#)
  - iterating over a channel, [Iterating over a Channel-Iterating over a Channel](#)
  - other flavors of channel, [Other flavors of Channel](#)
- runBlocking, [Your First Coroutine](#), [The async Coroutine Builder](#)
- runCatching, [Unhandled Versus Exposed Exceptions](#)
- runtime environment, [Android Runtime Environment](#)

## S

- Sample Java Methods, [Method tracing](#)

- Sample Method Trace, [Recording a sample method trace-Recording a sample method trace](#)
- sealed classes, [Sealed Classes](#)
- select expression, [The select Expression-The select Expression](#)
- self CPU time, [Analysis panel](#)
- send...(), [Looper/Handler](#)
- sequences, iterators versus, [Iterators Versus Sequences](#)
- Service, [Component context](#)
- services, [Services-Bound Services](#)
  - bound, [Bound Services-Bound Services](#)
  - started, [Started Services](#)
- SharedFlow
  - architecture, [The architecture](#)
  - buffer values, [Buffer values](#)
  - creating, [Create a SharedFlow](#)
  - as event bus, [Using SharedFlow as an Event Bus](#)
  - hot flows with, [Hot Flows with SharedFlow-An Example of StateFlow Usage](#)
    - (see also hot flows)
  - implementation, [The implementation-The implementation](#)
  - implementation testing, [Test of our implementation](#)
  - replay values, [Replay values](#)
  - sending values to, [Send Values to the SharedFlow](#)
  - streaming data with, [Using SharedFlow to Stream Data-Buffer values](#)
  - suspending/not suspending, [Suspend or not?](#)
- singleton objects, [Companion Objects](#)
- Stack, [Work Queues](#)
- started services, [Started Services](#)
- StateFlow, [StateFlow: A Specialized SharedFlow-An Example of StateFlow Usage](#)
- static functions, [Using Static Functions](#)
- streaming data, SharedFlow and, [Using SharedFlow to Stream Data-Buffer values](#)
- structured concurrency
  - app creation and, [Structured concurrency](#)
  - coroutines and (see structured concurrency with coroutines)

- parent-child relationship in, [The Parent-Child Relationship in Structured Concurrency](#)
- structured concurrency with coroutines, [A Quick Detour About Structured Concurrency-The Parent-Child Relationship in Structured Concurrency](#), [Structured Concurrency with Coroutines-Closing Thoughts](#)
  - automatic cancellation, [Automatic Cancellation](#)
  - cancellation, [Cancellation-Causes of Cancellation](#)
  - cancelling a task delegated to a third-party library, [Cancelling a Coroutine-Cancelling a Coroutine](#)
  - causes of cancellation, [Causes of Cancellation-Causes of Cancellation](#)
  - coroutine cancellation, [Cancelling a Coroutine-Cancelling a Coroutine](#)
  - coroutine lifecycle, [Coroutine Lifecycle-Job holds the state](#)
  - coroutines which are cooperative with cancellation, [Coroutines That Are Cooperative with Cancellation-Coroutines That Are Cooperative with Cancellation](#)
  - delay cancellation, [delay Is Cancellable](#)
  - exception handling, [Exception Handling-Unhandled Exceptions](#)
  - exposed exceptions, [Exposed Exceptions-Exposed Exceptions](#)
  - handling cancellation, [Handling Cancellation](#)
  - parallel decomposition, [Parallel Decomposition](#)
  - supervision, [Supervision-Supervision](#)
  - supervisorScope builder, [supervisorScope Builder](#)
  - suspending functions, [Suspending Functions-Summary of Suspending Functions Versus Traditional Threading](#)
  - unhandled exceptions, [Unhandled Exceptions-Unhandled Exceptions](#)
  - unhandled versus exposed exceptions, [Unhandled Versus Exposed Exceptions-Unhandled Versus Exposed Exceptions](#)
- subscribe method, [Error Handling](#)
- supervision, [Supervision-Supervision](#)
- SupervisorJob, [Supervision](#)
- supervisorScope, [supervisorScope Builder](#), [Exposed Exceptions](#)
- suspending functions, [Suspending Functions-Suspending Functions Under the Hood](#)

- and coroutines, [Using Suspending Functions and Coroutines-Using Suspending Functions and Coroutines](#)
- example, [Using Coroutines and Suspending Functions: A Practical Example-Using Coroutines and Suspending Functions: A Practical Example](#)
- HandlerThread, [A Reminder About HandlerThread-A Reminder About HandlerThread](#)
- java.util.concurrent.ExecutorService approach for, [Traditional Approach Using java.util.concurrent.ExecutorService-Traditional Approach Using java.util.concurrent.ExecutorService](#)
- lower-level constructs of, [Suspending Functions Under the Hood-Suspending Functions Under the Hood](#)
- structured concurrency with coroutines, [Suspending Functions-Summary of Suspending Functions Versus Traditional Threading](#)
- traditional threading versus, [Summary of Suspending Functions Versus Traditional Threading](#)
- synchronization, [Visibility](#), [Mutexes](#)
- system services, [System Services](#)

## T

- terminal operator, [An Introduction to Flows](#)
- ternary expression, [The Unit Type](#)
- thread activity timeline, [Thread activity timeline](#)
- thread managing tools, [Tools for Managing Threads-Executors and ExecutorServices](#)
  - Executors and ExecutorServices, [Executors and ExecutorServices-Executors and ExecutorServices](#)
  - Looper/Handler, [Looper/Handler-Looper/Handler](#)
- thread safety, [Thread Safety-Visibility](#), [Thread Safety-Summary](#)
  - atomicity, [Atomicity](#)
  - back pressure, [Back Pressure-Back Pressure](#)
  - blocking call versus nonblocking call, [Blocking Call Versus Nonblocking Call](#)
  - defined, [Concurrency in Android](#)
  - example of thread issue, [An Example of a Thread Issue-An Example of a Thread Issue](#)

- invariants, [Invariants-Thread-Safe Collections](#)
- mutexes, [Mutexes](#)
- thread confinement, [Thread Confinement](#)
- thread contention, [Thread Contention](#)
- thread-safe collections, [Thread-Safe Collections-Thread-Safe Collections](#)
- visibility, [Visibility](#)
- work queues, [Work Queues](#)
- Thread View, viewing network calls with, [Viewing network calls with Connection View and Thread View](#)
- thread, defined, [Concurrency in Android](#)
- Thread.setUncaughtExceptionHandler, [A Quick Detour About Structured Concurrency](#)
- Thread.sleep(), [View](#)
- threading model
  - basics, [The Android Threading Model](#)
  - limitations of, [Limitations of the Threading Model](#)
- threading, suspending functions versus, [Summary of Suspending Functions Versus Traditional Threading](#)
- ThreadLocal, [Thread Confinement](#)
- ThreadPoolExecutor, [Executors and ExecutorServices](#), [Limitations of the Threading Model](#), [Traditional Approach Using java.util.concurrent.ExecutorService](#)
- toString, [Data Classes](#)
- total CPU time, [Analysis panel](#)
- Trace Java Methods, [Method tracing](#)
- Trace System Calls, [Method tracing](#)
- transformation functions, [Kotlin Transformation Functions-Iterators Versus Sequences](#), [Map](#), [What Happens in Case of Error?](#)
  - Boolean functions, [The Boolean Functions](#)
  - filter functions, [Filter Functions-Map](#)
  - flatMap, [flatMap](#)
  - grouping, [Grouping](#)
  - iterators versus sequences, [Iterators Versus Sequences](#)
- TrekMe, [Performance Considerations with Android Profiling Tools](#), [Android Profiler](#)
- try/catch block, [The try/catch Block-The try/catch Block](#)



- tryEmit, [Send Values to the SharedFlow](#), [Suspend or not?](#)
- type inference, [Null Safety](#)
- type system, [The Kotlin Type System-Generics](#)
  - function types, [Function Types-Function Types](#)
  - generics, [Generics](#)
  - null safety, [Null Safety-Null Safety](#)
  - primitive types, [Primitive Types](#)
  - unit type, [The Unit Type](#)

## U

- UncaughtExceptionHandler, [Unhandled Exceptions](#)
- unhandled exceptions
  - about, [Unhandled Exceptions-Unhandled Exceptions](#)
  - exposed versus, [Unhandled Versus Exposed Exceptions-Unhandled Versus Exposed Exceptions](#)
- Unit, [The Unit Type](#)
- unit type, [The Unit Type](#)
- unlimited channels, [Unlimited Channel-Unlimited Channel](#)
- upper bound type, [Creating Containers](#)
- upstream exceptions, [Exception transparency](#)
- upstream flow, [Use Case #1: Interface with a Callback-Based API](#)

## V

- View, [MVC: The Foundation](#)
- ViewModel, [Memory Leaks](#), [View-Model](#), [View](#), [Detecting Memory Leaks with LeakCanary](#)
- @ViewModelInject, [Reducing Unnecessary Work](#)
- viewModelScope, [Suspending Functions Under the Hood](#), [Using Suspending Functions and Coroutines](#)
- visibility, [Visibility](#)
- visibility modifiers, [Visibility Modifiers-Visibility Modifiers](#)
- VSync, [Reducing Programmatic Draws with Drawables](#)

## W

- whileSelect, [Use Case #3: Create a Custom Operator](#)

- widgets, [Widgets](#)
- withContext, [Using Suspending Functions and Coroutines](#)
- work queues, [Work Queues](#)
- WorkerPool, [Thread Contention](#)
- WorkManager, [WorkManager](#)

## Y

- yield(), [Cancelling a Coroutine](#), [delay Is Cancellable](#)

## Z

- Zygote, [The Android Application Environment](#)

[Support](#) | [Sign Out](#)

©2022 O'REILLY MEDIA, INC. [TERMS OF SERVICE](#) | [PRIVACY POLICY](#)