

Android with Kotlin Bootcamp

Day 4 - Performing Background Work



Nate Ebel

Android Developer & Instructor

@n8ebel goobar.dev



Any questions from yesterday?

What are we
learning today?

Saving local data using Room
MVVM Architecture with Android Jetpack
Async programming constructs
Kotlin Coroutines
Making Network Requests



Let's check in on our “want to learns”

Any Questions?

Using Room for Local Database



Nate Ebel

Android Developer & Instructor

@n8ebel goobar.dev

Overview

Understand what Room is

Understand how to define a Room entity

Understand how to save data

Understand how to query data

Understand how to delete data

Why use Room?

Why Room?

Abstraction over SQLite

Full power of SQLite

Generate tables and interaction code via annotations

Key Room Components

Database

Entities

DAO

```
@Entity  
data class User(  
    @PrimaryKey val uid: Int,  
    @ColumnInfo(name = "first_name") val firstName: String?,  
    @ColumnInfo(name = "last_name") val lastName: String?  
)
```

Define an Entity Table

Adding the `@Entity` annotation will generate a table corresponding to the annotated class

```
@Dao
interface UserDao {
    @Query("SELECT * FROM user")
    fun getAll(): List<User>

    @Query("SELECT * FROM user WHERE uid IN (:userIds)")
    fun loadAllByIds(userIds: IntArray): List<User>

    @Query("SELECT * FROM user WHERE first_name LIKE :first AND " +
        "last_name LIKE :last LIMIT 1")
    fun findByName(first: String, last: String): User

    @Insert
    fun insertAll(vararg users: User)

    @Delete
    fun delete(user: User)
}
```

Define a Data Access Object (DAO)

Annotate an interface with `@Dao` to convert interface methods into database interactions

```
@Database(entities = arrayOf(User::class), version = 1)
abstract class AppDatabase : RoomDatabase() {
    abstract fun userDao(): UserDao
}
```

Generate a Database

Annotate an abstract class with `@Database` - specifying supported `@Entity` types and any required `@Dao` interfaces

Room Interactions

- Synchronous queries must be done off the main thread**
- Supports RxJava**
- Supports LiveData**
- Supports Kotlin coroutines**

MVVM with Android Architecture Components



Nate Ebel
Android Developer & Instructor

@n8ebel goobar.dev

Overview

- Understand the need for app architecture**
- Understand common architectural patterns**
- Understand Android Architecture Components**
- Understand Android ViewModel**
- Understand LiveData**

Why is application architecture
important?

Text Chunking: Four Items

MVC

MVP

MVVM

MVI

Text Chunking: Four Items

Model-View-Controller

Model-View-Presenter

Model-View-ViewModel

Model-View-Intent

Model-View-Controller

Controller accepts user interaction, which updates the model, which updates the view that the user sees

<https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93presenter>

Model-View-Presenter

Active presenter coordinates all interactions between the model/data layer and the view layer

<https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93presenter>

Model-View-ViewModel

ViewModel and Model interact while data from the ViewModel is bound to the View

<https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93presenter>

Model-View-ViewModel

ViewModel and Model interact while data from the ViewModel is bound to the View

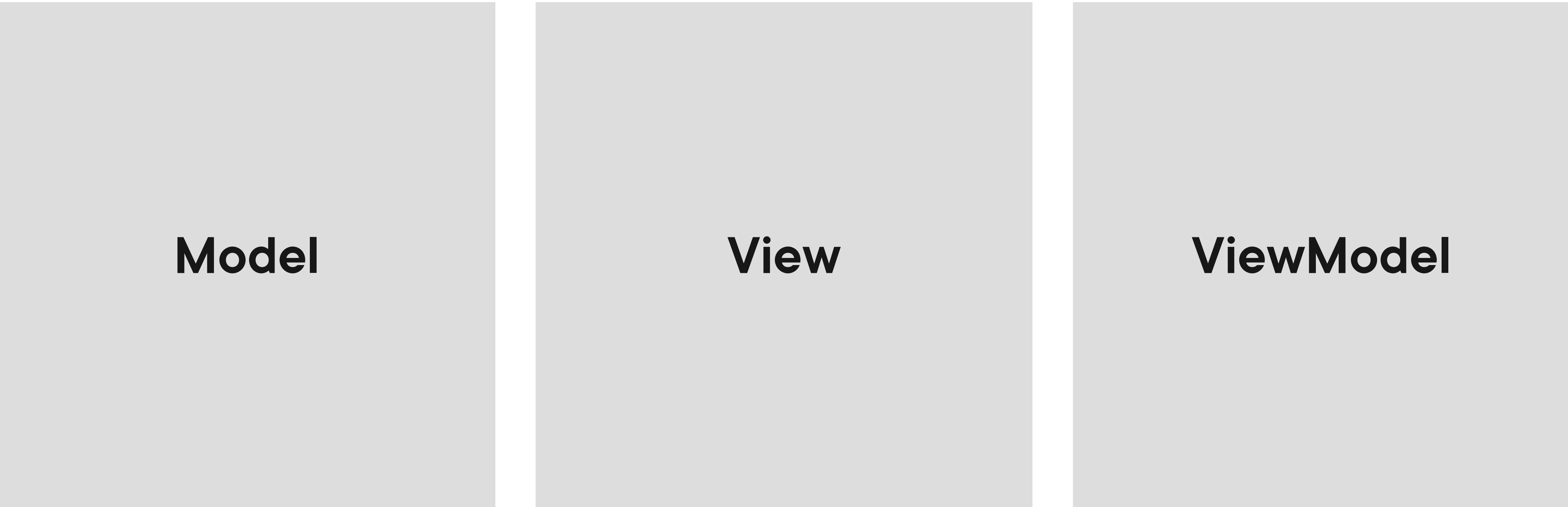
<https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93viewmodel>

Model-View-Intent

An implementation of unidirectional data flow in which immutable models represent the current state of the UI, while intents representing desired actions are generated through View interactions and reduced/processed by some controller which converts those intents back into new model states.

<https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93viewmodel>

MVVM on Android



The diagram illustrates the MVVM pattern on Android. It consists of three light gray rectangular boxes arranged horizontally. The first box on the left is labeled "Model". The second box in the middle is labeled "View". The third box on the right is labeled "ViewModel".

Model

View

ViewModel

MVVM on Android

Repository/DB

Activity/Fragment

ViewModel

Android Architecture Components



ViewModel



LiveData

Android Architecture Components

ViewModel

Designed to store and manage UI-related data in a lifecycle conscious way. The ViewModel class allows data to survive configuration changes such as screen rotations.

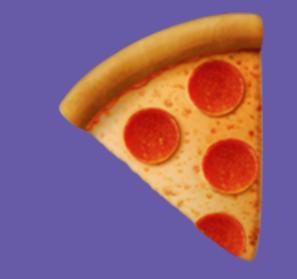
<https://developer.android.com/topic/libraries/architecture/viewmodel>

Android Architecture Components

LiveData

Observable data holder class. Unlike a regular observable, LiveData is lifecycle-aware, meaning it respects the lifecycle of other app components, such as activities, fragments, or services.

<https://developer.android.com/topic/libraries/architecture/livedata>



Lunch

Writing Multithreaded Applications



Nate Ebel

Android Developer & Instructor

@n8ebel goobar.dev

Overview

Understand the Android threading model

Understand both short, and long-running tasks

Understand Threads, Executors, and AsyncTask

Explore RxJava and Coroutines

Understand Services and WorkManager

Main Thread

The default thread created for an application. It's the thread on which all UI updates/interactions happen and on which components are instantiated.

<https://developer.android.com/guide/components/processes-and-threads>

Blocking the Main Thread



- Degraded performance
- UI Jank
- Activity Not Responding dialog
- Crashes

Kinds of Work

Short-running

Load data from a local database

**Processing a collection of local
model objects**

**Making a network request for a small
JSON response**

Long-running

Downloading a large file

Uploading a large file

Building a machine learning model

Async Programming Constructs

Threads

Executors

AsyncTask

RxJava

Coroutines

Thread

- A thread of execution for a program
- May have multiple Threads running at once
- Extend a Thread directly
- Pass a Runnable to a thread
- Threads require memory and cpu resources

Handler

Allows submission of Runnables that will run on the Thread in which the Handler was created.

Citation: Author/Source, Title, Link/Short Url

ThreadPoolExecutor

ThreadPools restrict the number of Threads
Executor allows Runnable submission
ThreadPoolExecutor provides submission of Runnables which are then run across the pool of available threads
Efficient
Inconvenient to work with

AsyncTask



- Originally very widely used in Android
- An abstraction over Thread and Handler
- Had a number of problems
- Deprecated and no longer recommended

Coroutines

Kotlin's default async primitives

Conceptually similar to “lightweight thread”

Asynchronously run code without blocking

Support structured concurrency

Long Running and Repeated Work



Service



WorkManager

Service

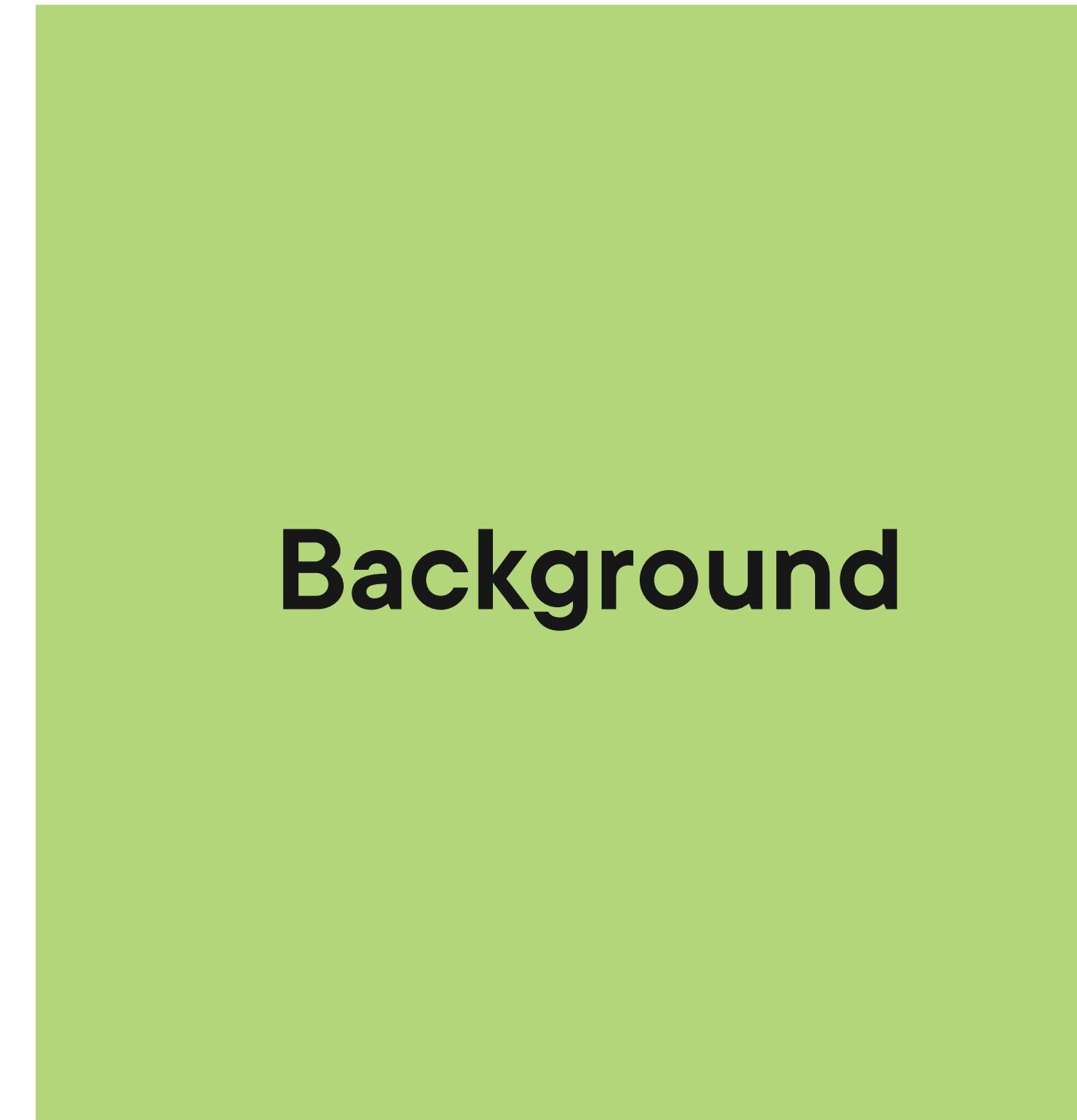
Core Android component for performing long-running tasks.

<https://developer.android.com/guide/components/services>

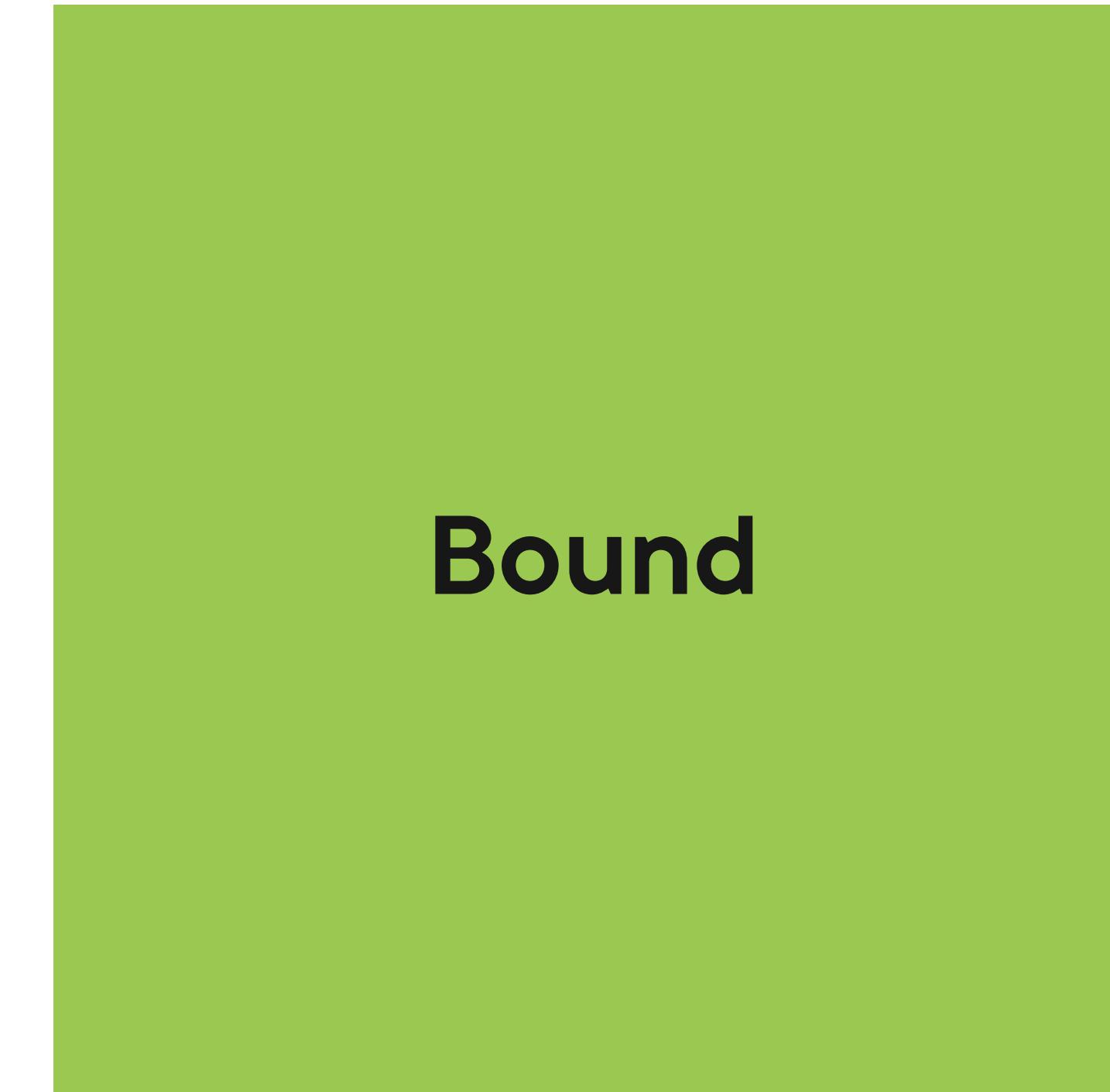
Service Types



Foreground



Background



Bound

Foreground Service

Performs work that is noticeable to the user, and will keep your Application active even if your UI is in the background.

<https://developer.android.com/guide/components/services>

Background Service

Performs work that is not noticeable to the user. Restrictions apply.

<https://developer.android.com/guide/components/services>

Bound Service

Support client/server interaction and interprocess communication with Application components.

<https://developer.android.com/guide/components/services>

When to use a Service?

WorkManager

Schedules reliable, configurable, asynchronous tasks that work consistently across Android API versions.

<https://developer.android.com/topic/libraries/architecture/workmanager>

WorkManager

Abstraction over multiple components such as JobScheduler and AlarmManager

WorkConstraints

Scheduling

Retry

Chaining

When to use WorkManager?

Kotlin Coroutines



Nate Ebel

Android Developer & Instructor

@n8ebel goobar.dev

Overview

Understand what a coroutine is

Understand structured concurrency

Understand how to work with suspending functions

Explore error handling

Explore StateFlow and SharedFlow

Coroutine

A suspendible computation - conceptually similar to a thread, but not bound to a single thread and supporting the concept of structured concurrency.

<https://kotlinlang.org/docs/coroutines-basics.html#structured-concurrency>

Structured Concurrency

Ensures that computations are not lost, and do not leak. Enforces a scoped hierarchy to coroutines that are created.

<https://kotlinlang.org/docs/coroutines-basics.html#structured-concurrency>

Coroutine Scope

Controls the lifecycle of a coroutine. Avoids leaks, and supports cancellation. Enables scoping of asynchronous executions to the scope/lifecycles of other objects.

Coroutine Builders



- `runBlocking{}`
- `withContext{}`
- `launch{}`
- `async{}`

Error Handling

Cancelled coroutines throw
CancellationException

Errors propagate to siblings and to parent context

try/catch

CoroutinesExceptionHandler

Flow

Represents a *cold* stream of asynchronously computed values

<https://kotlinlang.org/docs/flow.html#flows>

Types of Flow



StateFlow



SharedFlow

StateFlow

State-holder observable flow that emits the current and new state updates to its collectors. In Android, StateFlow is a great choice to maintain an observable UI state from a ViewModel.

<https://developer.android.com/kotlin/flow/stateflow-and-sharedflow>

SharedFlow

A hot flow that emits values to all consumers that collect from it. A great choice for emitting events to many possible consumers.

<https://developer.android.com/kotlin/flow/stateflow-and-sharedflow>

Making Network Requests with Volley



Nate Ebel

Android Developer & Instructor

@n8ebel goobar.dev

Overview

Understand basic networking concepts

Explore Android networking tools

**Understand how to make a network request
using Volley**

Networking on Android

HttpURLConnection

Retrofit

Volley

Networking on Android



Volley

Volley

Volley is an HTTP library that makes networking for Android apps easier and most importantly, faster.

<https://developer.android.com/training/volley>

Volley

- Supports common request types out of the box
- Disk & memory response caching
- Support for retry and backoff
- Cancellation support



Office Hours