Starting at 10am PST

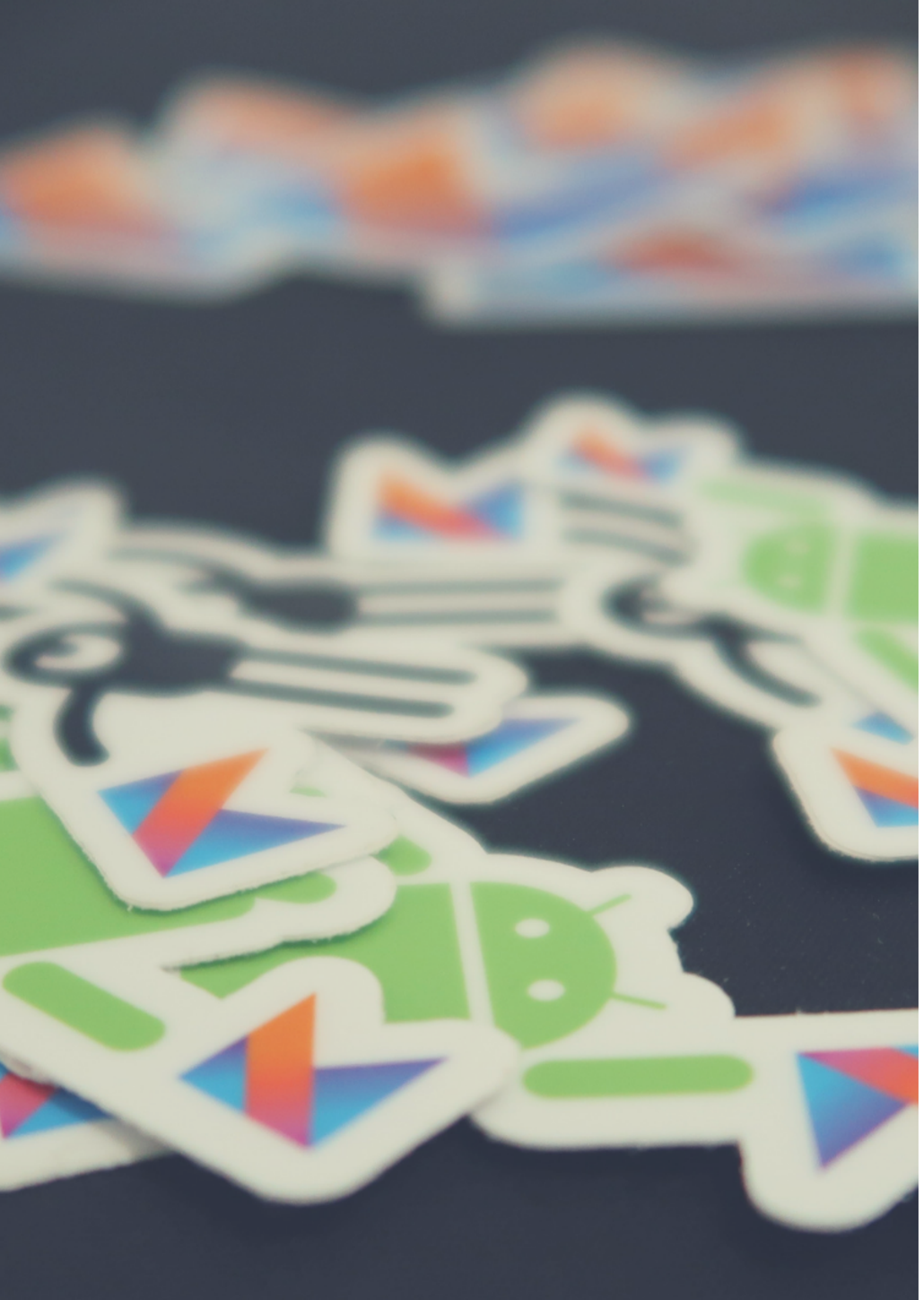
Due to an All Hands at 9am PST, Day 3 of our Android Bootcamp will be starting an hour later at 10am PST

See you then 👋

Module Title in Title Case

Author Name
Author Title in Title Case

@authortwitter www.authorsite.com



Any questions from yesterday?

What are we learning today?

Explicit & Implicit Intents

RecyclerView

Local Storage Solutions

DataStore

Building for a global audience

MVVM & Android Architecture Components

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

Any Questions?

Working with Intents

Nate Ebel

Android Developer & Instructor

@n8ebel goobar.dev

Overview

- Understand what Intents are used for
- Understand Implicit vs Explicit Intents
- Understand how to start an Activity with an Explicit Intent
- Understand how to perform an action with an Implicit Intent

Intent

A messaging object you can use to request an action from another app component

https://developer.android.com/guide/components/intents-filters

Intent Types

Explicit Intents

Implicit Intents

Explicit Intent

Specify the specific application or component to start - typically within your own app.

ex. “start our MainActivity”

https://developer.android.com/guide/components/intents-filters

Explicit Intent

Always delivered to its target, regardless of any intent filters the component declares.

https://developer.android.com/guide/components/intents-filters#Receiving

Implicit Intent

Specify an action to perform - and let the OS and user choose which application to handle the request based on declared intent filters.

ex. “send a text”

https://developer.android.com/guide/components/intents-filters

Intent Filter

Specifies the type of intents it accepts based on the intent's action, data, and category. The system delivers an implicit intent to your app component only if the intent can pass through one of your intent filters

https://developer.android.com/guide/components/intents-filters#Receiving

What are some Implicit Intent examples?

Displaying List Data with RecyclerView

Nate Ebel

Android Developer & Instructor

@n8ebel goobar.dev

Overview

- Understand the need for efficient data display
- Understand the efficiencies of RecyclerView
- Understand how to display data with RecyclerView
- Understand efficient list diffing

Why is efficient display of list data so important?

List Display Widgets

ListView

RecyclerView

List Display Widgets

RecyclerView

RecyclerView

Makes it easy to efficiently display large sets of data. You provide the data and define how each item layout, and the RecyclerView library dynamically creates the elements when they're needed.

https://developer.android.com/guide/topics/ui/layout/recyclerview

RecyclerView

RecyclerView “recycles” individual elements. When an item scrolls off the screen, RecyclerView doesn't destroy the item’s view. Instead, RecyclerView reuses the view for new items that have scrolled on to the screen.

https://developer.android.com/guide/topics/ui/layout/recyclerview

RecyclerView Components

RecyclerView

RecyclerView.ViewHolder

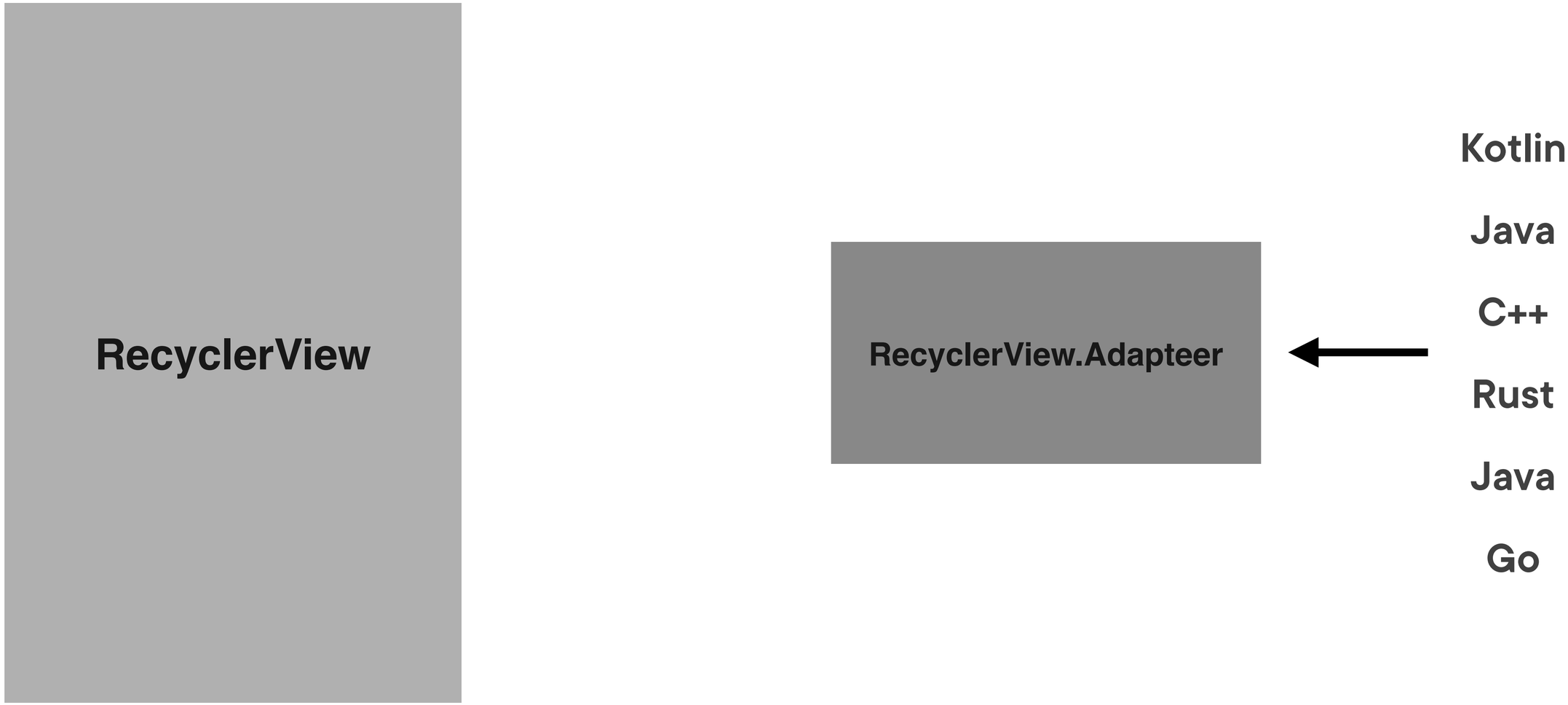
RecyclerView.Adapter

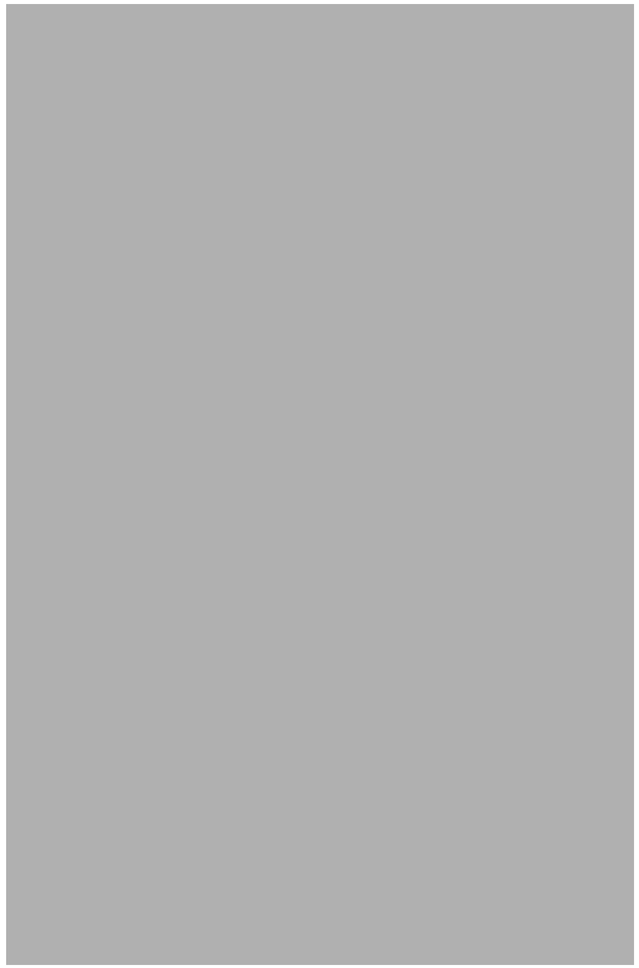
LayoutManger

RecyclerView

RecyclerView

RecyclerView.Adapteer


RecyclerView
RecyclerView.Adapteer
Kotlin
Java
C++
Rust
Java
Go



RecyclerView.Adapteer

Kotlin

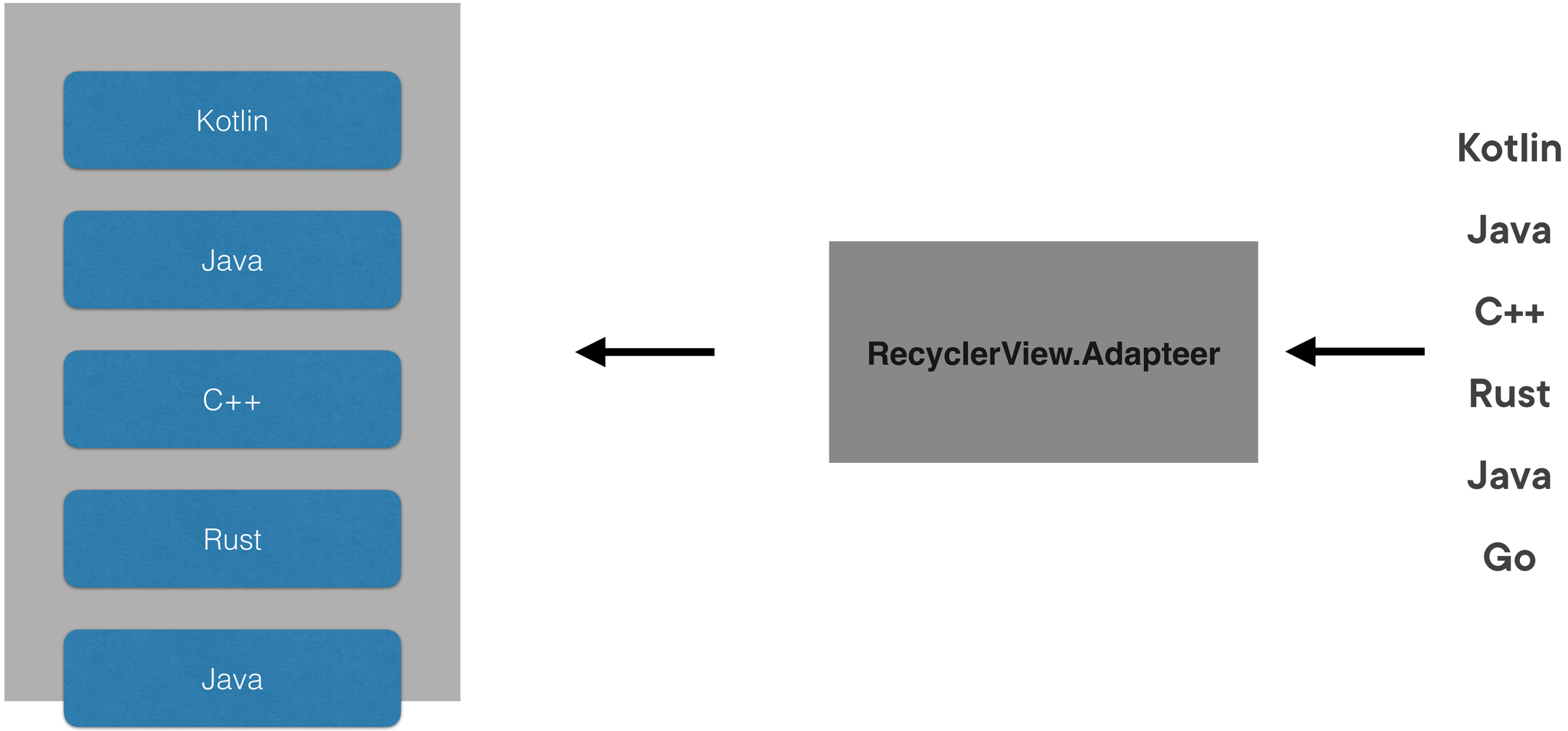
Java

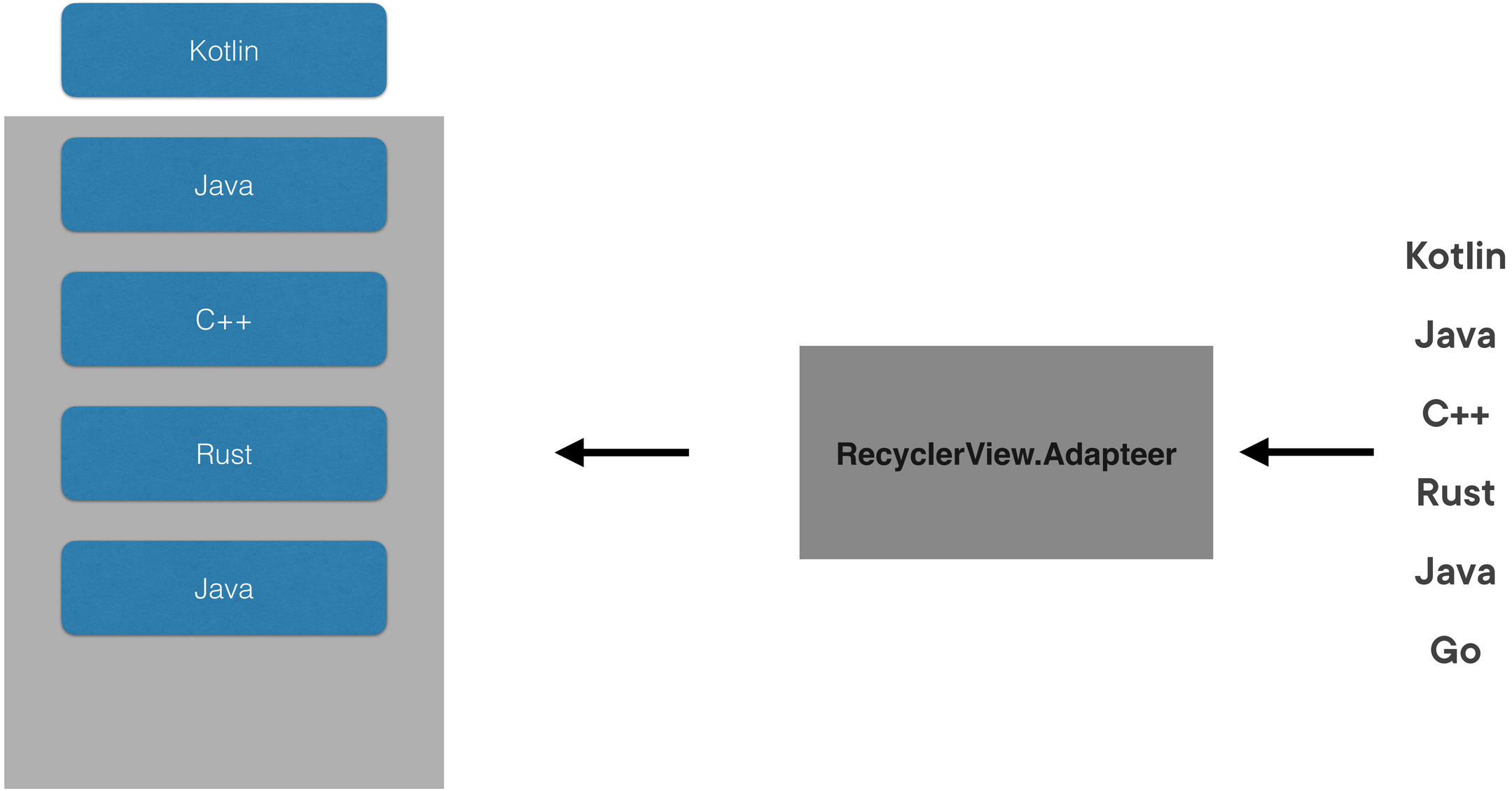
C++

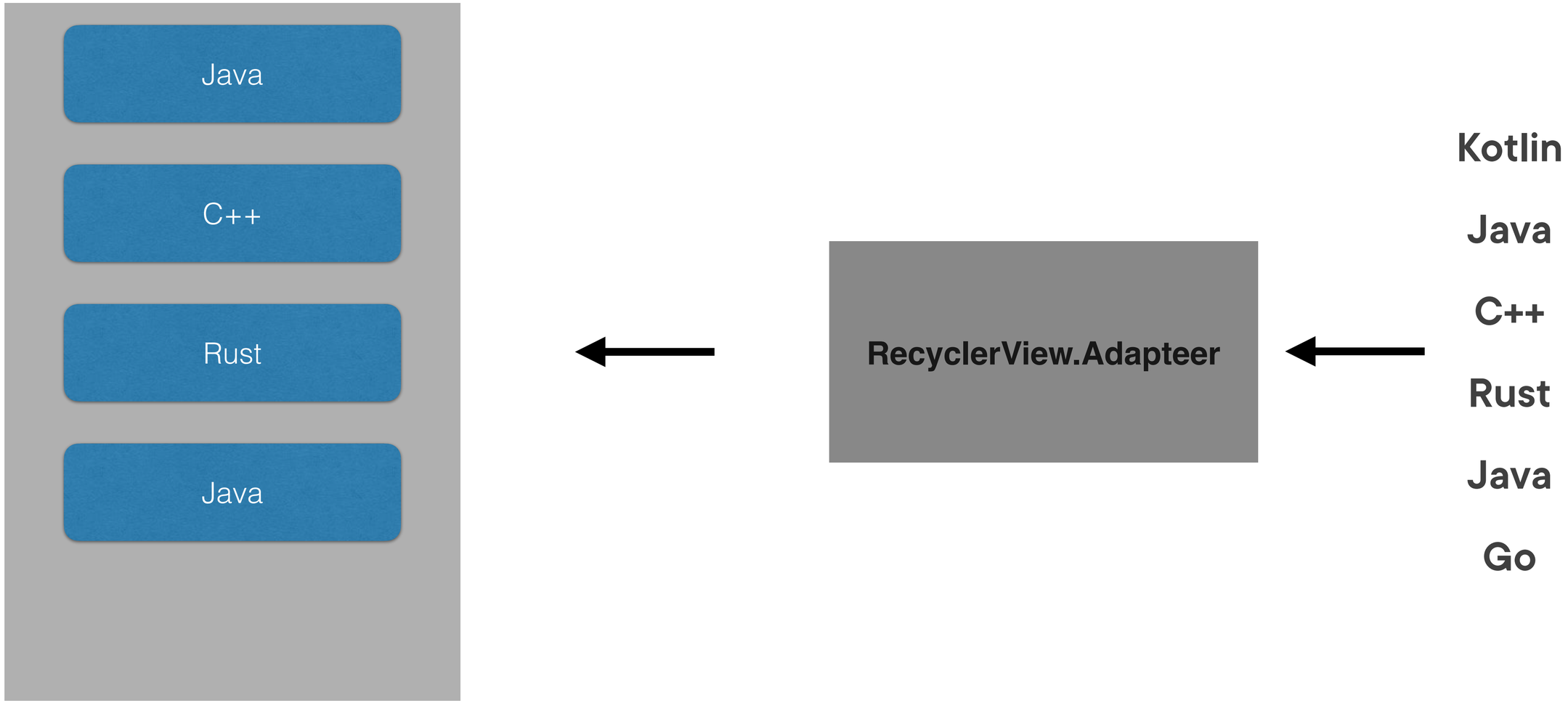
Rust

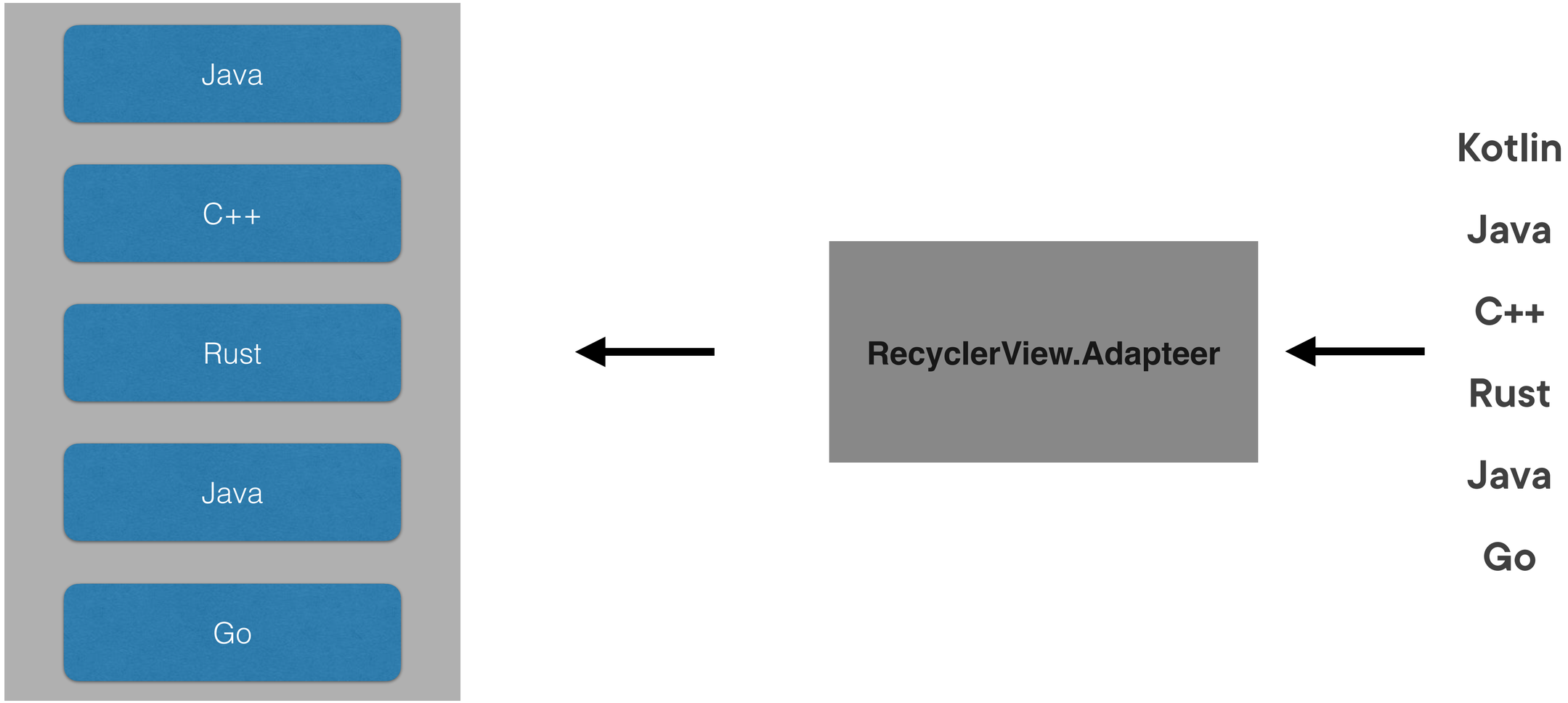
Java

Go


Kotlin
Java
C++
Rust
Java
RecyclerView.Adapteer
Kotlin
Java
C++
Rust
Java
Go


Kotlin
Java
C++
Rust
Java
RecyclerView.Adapteer
Kotlin
Java
C++
Rust
Java
Go


Java
C++
Rust
Java
RecyclerView.Adapteer
Kotlin
Java
C++
Rust
Java
Go


Java
C++
Rust
Java
Go
RecyclerView.Adapteer
Kotlin
Java
C++
Rust
Java
Go

RecyclerView

RecyclerView is the ViewGroup

RecyclerView.ViewHolder controls item layout

RecyclerView.Adapter converts data into ViewHolders

LayoutManger controls how item views are arranged

ListAdapter

Base class for presenting List data in a RecyclerView, including computing diffs between Lists on a background thread.

https://developer.android.com/reference/androidx/recyclerview/widget/ListAdapter

Exploring Local Storage Solutions

Nate Ebel

Android Developer & Instructor

@n8ebel goobar.dev

Overview

Understand the need for local data storage

Understand SharedPreferences

Understand DataStore

Understand Room

Why do we need to save local data?

What data are we saving?

Application settings/preferences

Profile data

User generated content

Data fetched from network

Downloaded files

Local Storage Options

Filesystem

SharedPreferences

Jetpack DataStore

SQLite

Jetpack Room

ORMs

SharedPreferences

Abstraction interface for saving/accessing key/value data to a local file

https://developer.android.com/reference/android/content/SharedPreferences

SharedPreferences

- Great for key/value pairs of simple data types
- Data saved to local files
- Edits are made through an Editor
- DefaultSharedPreferences
- Named preferences
- PreferenceScreen for preference-generated UI for settings

DataStore

Allows you to store key-value pairs or typed objects with protocol buffers. DataStore uses Kotlin coroutines and Flow to store data asynchronously, consistently, and transactionally.

https://developer.android.com/topic/libraries/architecture/datastore

DataStore

- Modern replacement for SharedPreferences
- Key/value data
- Typed data via protocol buffers
- Efficient asynchronous edits

Protocol Buffers

Google's language-neutral, platform-neutral, extensible mechanism for serializing structured data – think XML, but smaller, faster, and simpler

https://developers.google.com/protocol-buffers

```
// example.proto

message Person {

  string name = 1;
  int32 id = 2;  // Unique ID number for this person.
  string email = 3;

  enum PhoneType
    MOBILE = 0;
    HOME = 1;
    WORK = 2;
  }

  message PhoneNumber {
    string number = 1;
    PhoneType type = 2;
  }
}
```

- **Defined in a .proto file**
- **Platform/language agnostic syntax**
- **Tooling translates this into source code for desired language such as C++, Java, Kotlin, etc**

Room

Provides an abstraction layer over SQLite to allow for more robust database access while harnessing the full power of SQLite.

https://developer.android.com/jetpack/androidx/releases/room


Lunch

Using DataStore for Local Storage

Nate Ebel

Android Developer & Instructor

@n8ebel goobar.dev

Overview

Understand how to use DataStore

Understand how to save data with DataStore

Understand how to retrieve data with DataStore

Understand how ti display local data

DataStore or SharedPreferences?

Which DataStore to Use?

Preferences DataStore

Proto DataStore

Preference DataStore

Stores and accesses data using keys. This implementation does not require a predefined schema, and it does not provide type safety.

https://developer.android.com/topic/libraries/architecture/datastore

Proto DataStore

Stores data as instances of a custom data type. This implementation requires you to define a schema using protocol buffers, but it provides type safety.

https://developer.android.com/topic/libraries/architecture/datastore

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

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

💡 Office Hours