



Android Development

Bas Suphakrit Nakhabat



Topic

- Tools and project structure
- Resources and View Layout
- Basic Kotlin
- Activity and Fragment
- Android Jetpack
- Network (OkHttp, Retrofit, Gson)
- Kotlin Flows
- Save key-value data
- Dependency Injection (optional)



Tools and project structure



Android Studio provides the fastest tools for building apps on every type of Android device.



Tools: Gradle

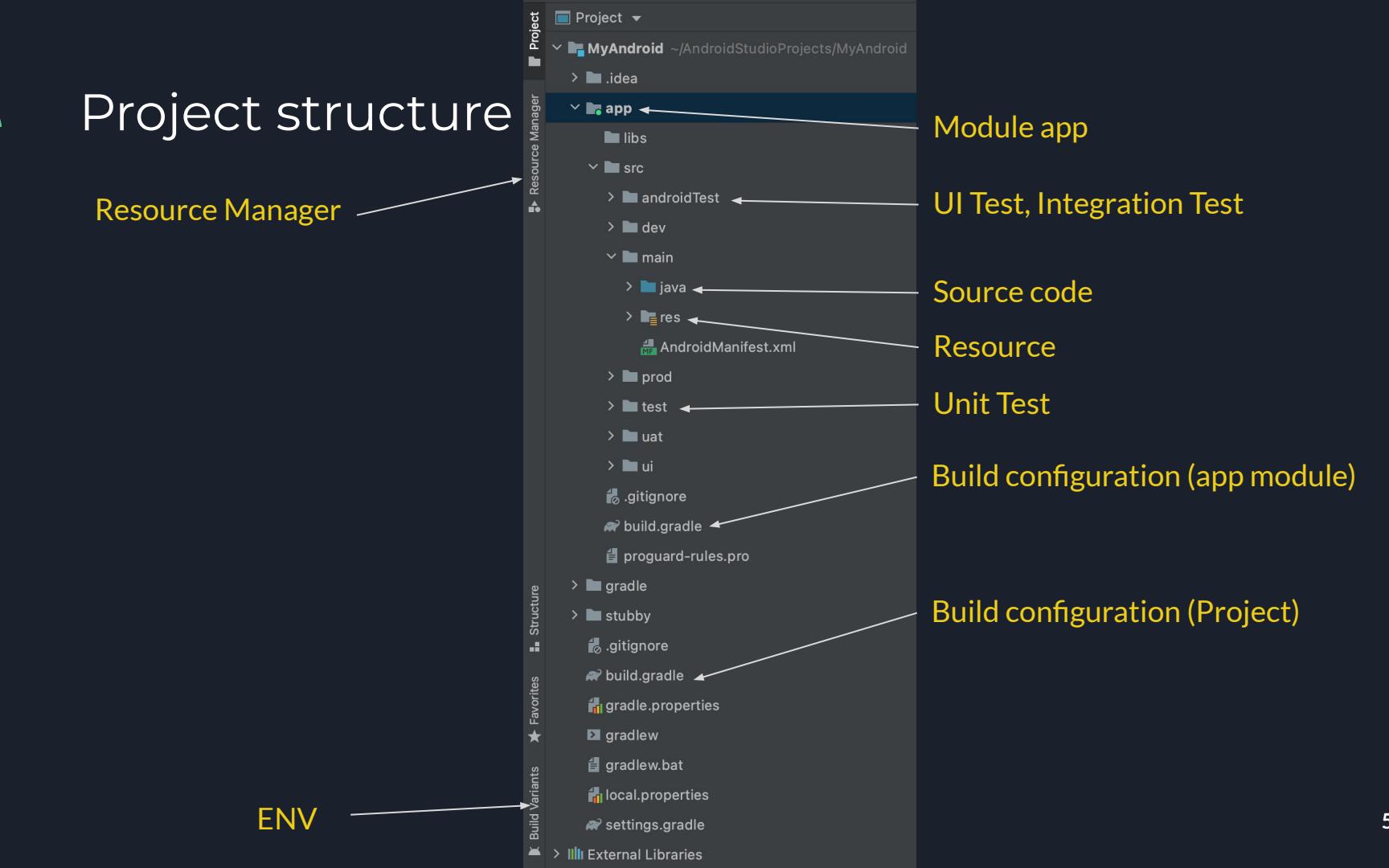


Gradle is an open-source build automation tool focused on flexibility and performance. Gradle build scripts are written using a Groovy or Kotlin DSL.



Project structure

Resource Manager



ENV



MyAndroid > app > src > main > java > com > example > myandroid > ui > MainActivity >

Project

MyAndroid ~/AndroidStudioProjects/MyAndroid

> .gradle

> .idea

> app

> build

libs

> src

> androidTest

> dev

> main

> java

> res

AndroidManifest.xml

> prod

> test

> uat

> ui

.gitignore

build.gradle

proguard-rules.pro

> build

> gradle

> stubby

.gitignore

build.gradle

gradle.properties

gradlew

gradlew.bat

Structure

Favorites

Build Variants

App

Build

+

-

x

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?

?



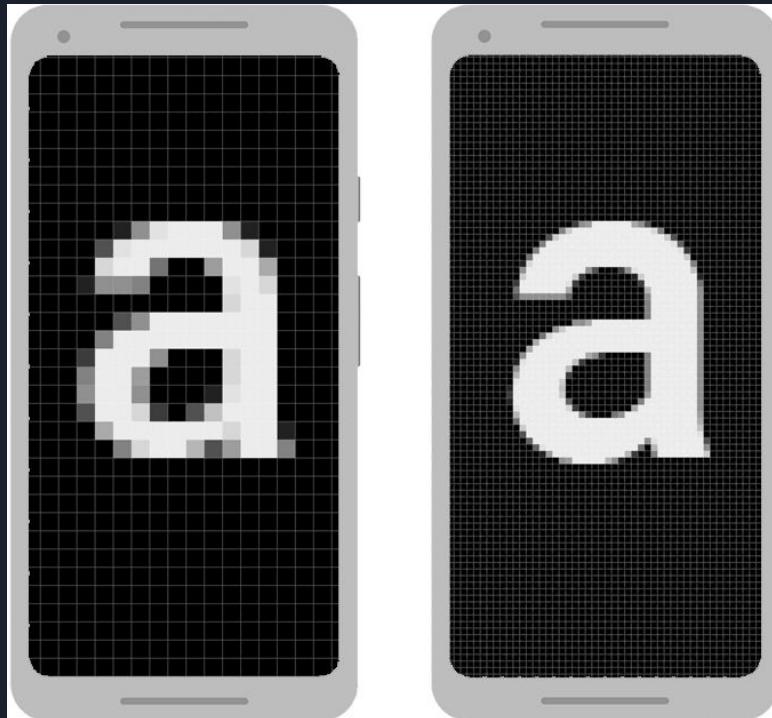
Resources

```
MyProject/  
    src/...  
        MyActivity.kt  
    res/  
        anim/  
            fade_in.xml  
            fade_out.xml  
        color/  
            text_color_state.xml  
        drawable/  
            graphic.png  
        font/  
            font_kanit.ttf  
    layout/  
        activity_main.xml  
        fragment_info.xml  
    mipmap/  
        icon_launcher.png
```

```
MyProject/  
    src/...  
        MyActivity.kt  
    res/  
        ...  
        values/  
            dimens.xml  
            strings.xml  
            styles.xml  
            colors.xml  
        values-th/  
            strings.xml
```



Support different pixel densities



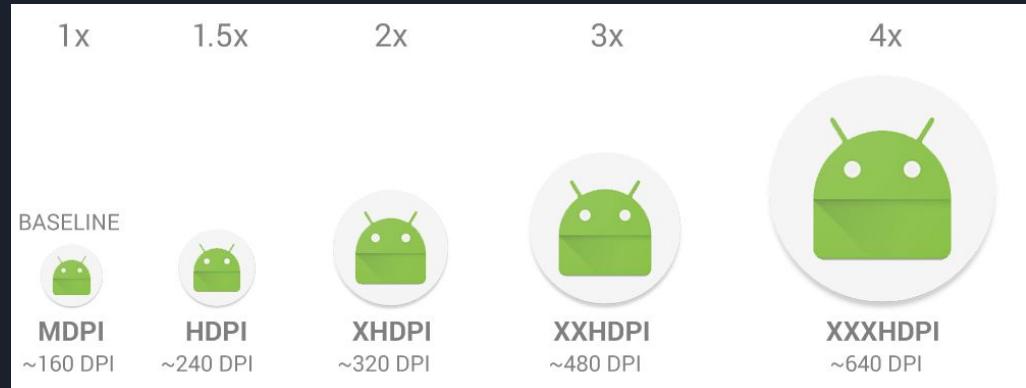
Two screens of the same size may have a different number of pixels

Spacing between two views, use **density-independent pixels (dp)**

Specifying text size, always use **scalable pixels (sp)**



Support different pixel densities



`res/`

`drawable-xxxhdpi/`
`awesome-image.png`
`drawable-xxhdpi/`
`awesome-image.png`
`drawable-xhdpi/`
`awesome-image.png`
`drawable-hdpi/`
`awesome-image.png`
`drawable-mdpi/`
`awesome-image.png`

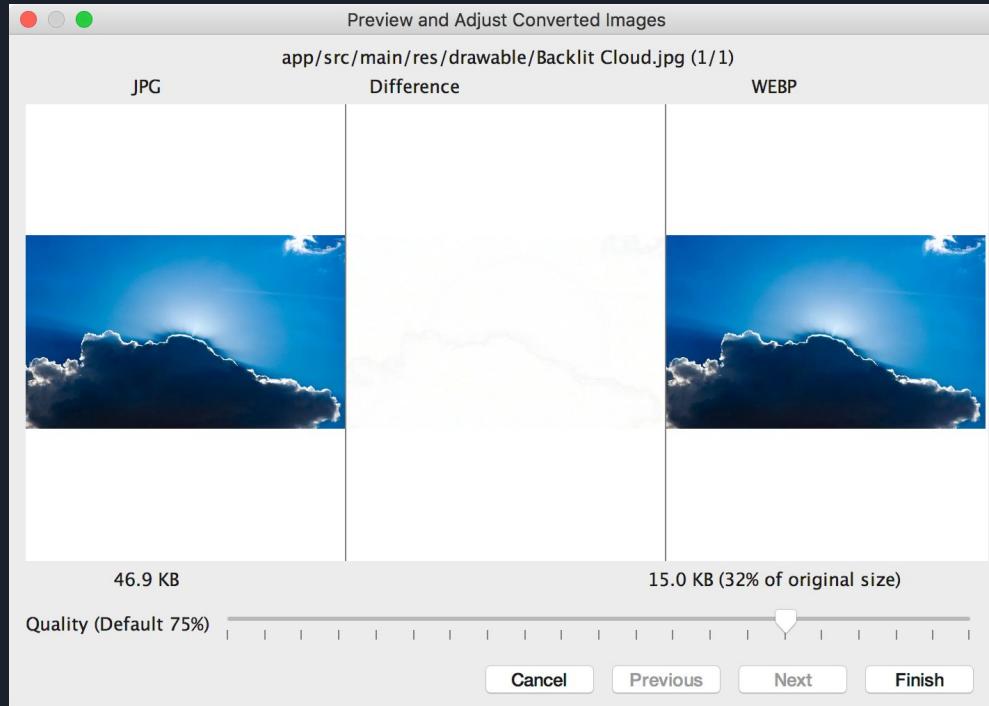


Convert images to WebP

WebP is an image file format from Google that provides lossy compression (like JPEG) as well as transparency (like PNG) but can provide better compression than either JPEG or PNG. Lossy WebP images are supported in Android 4.0 (API level 14) and higher, and lossless and transparent WebP images are supported in Android 4.3 (API level 18) and higher. This page shows how to convert images to WebP format and how to convert WebP images to PNG format.



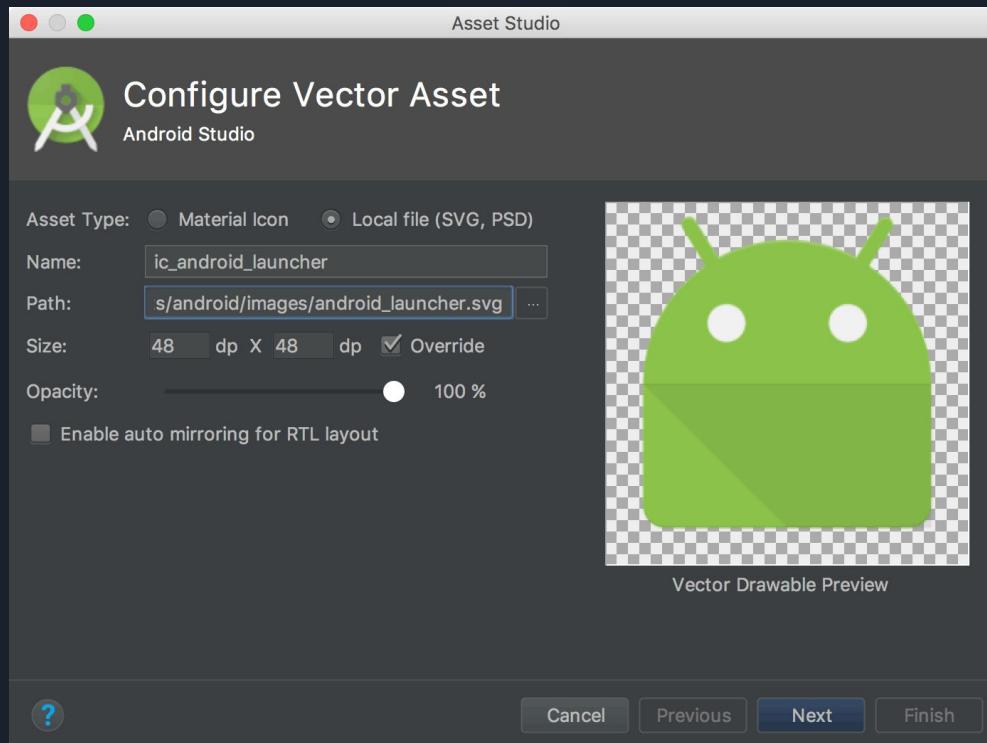
Convert images to WebP



<https://developer.android.com/studio/write/convert-webp>

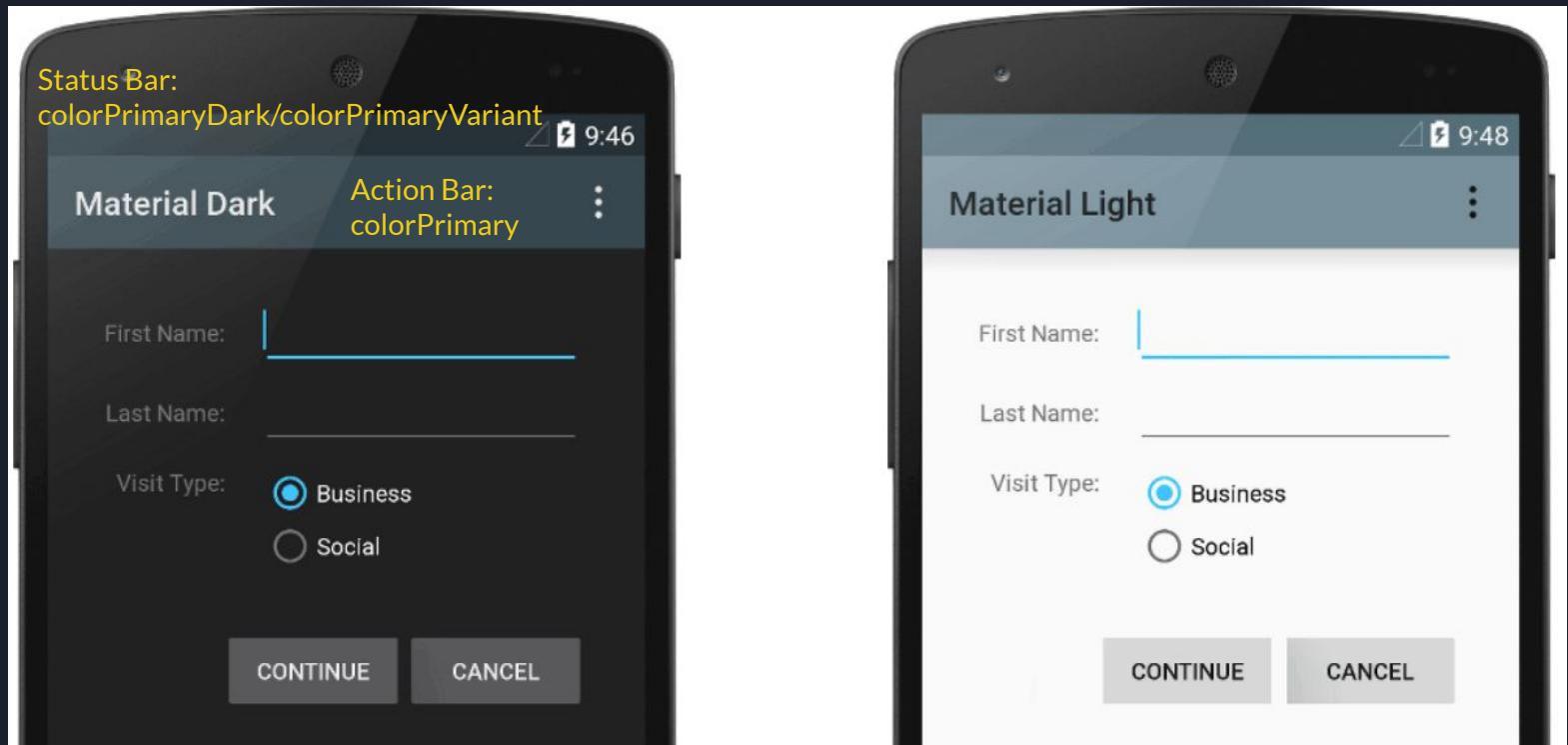


Use vector graphics instead



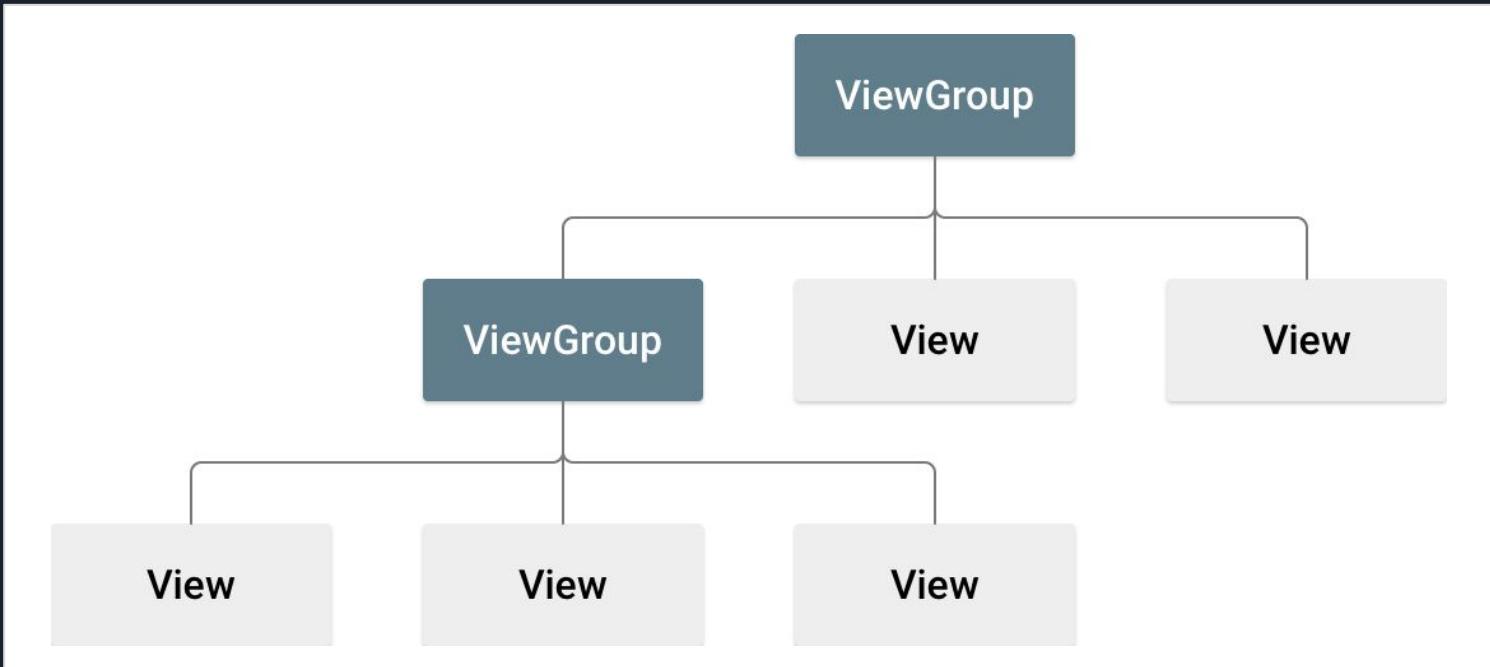


Styles and Themes



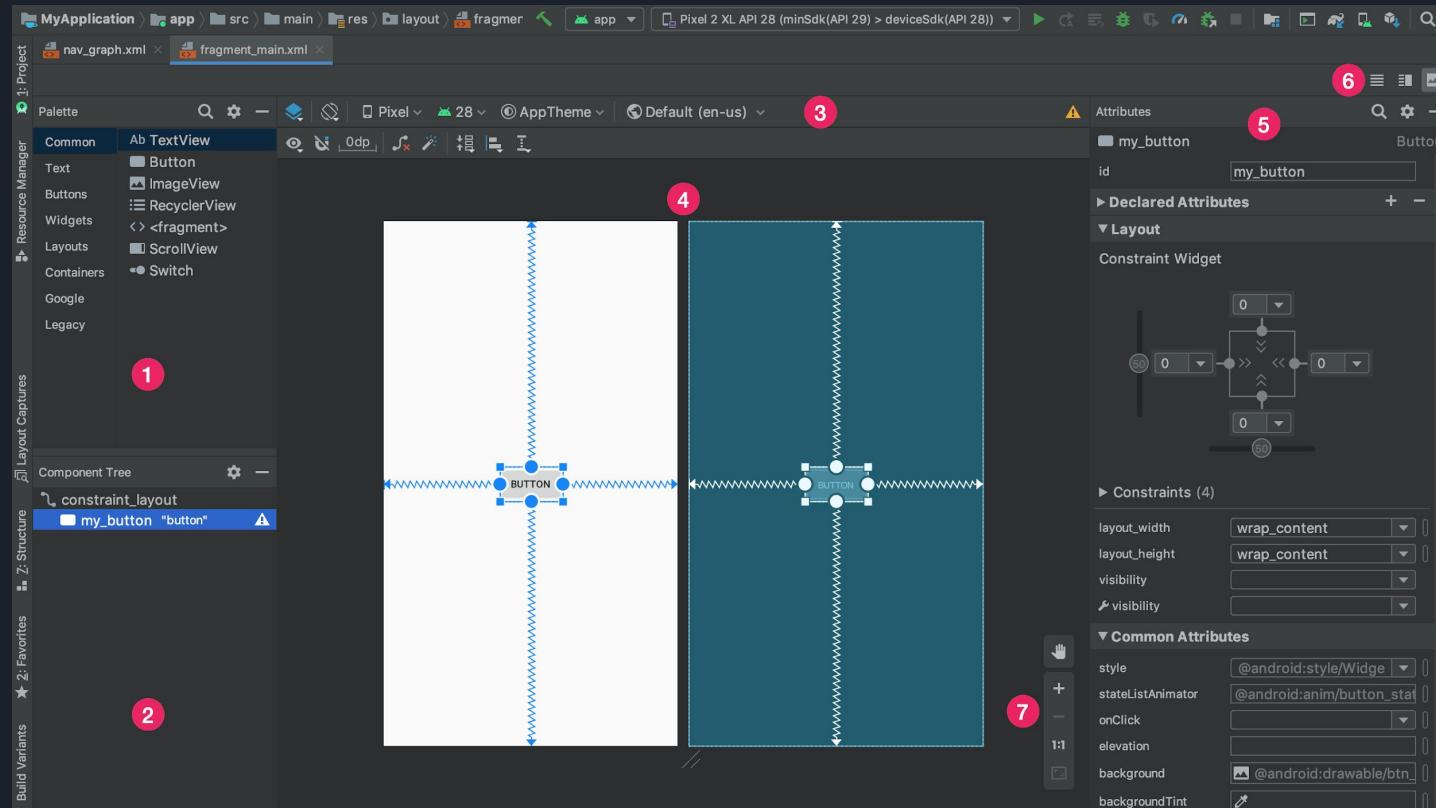


Layout





Layout: Editor





Layout: Editor

The screenshot shows the Android Layout Editor interface with a button component selected. The editor is divided into several sections:

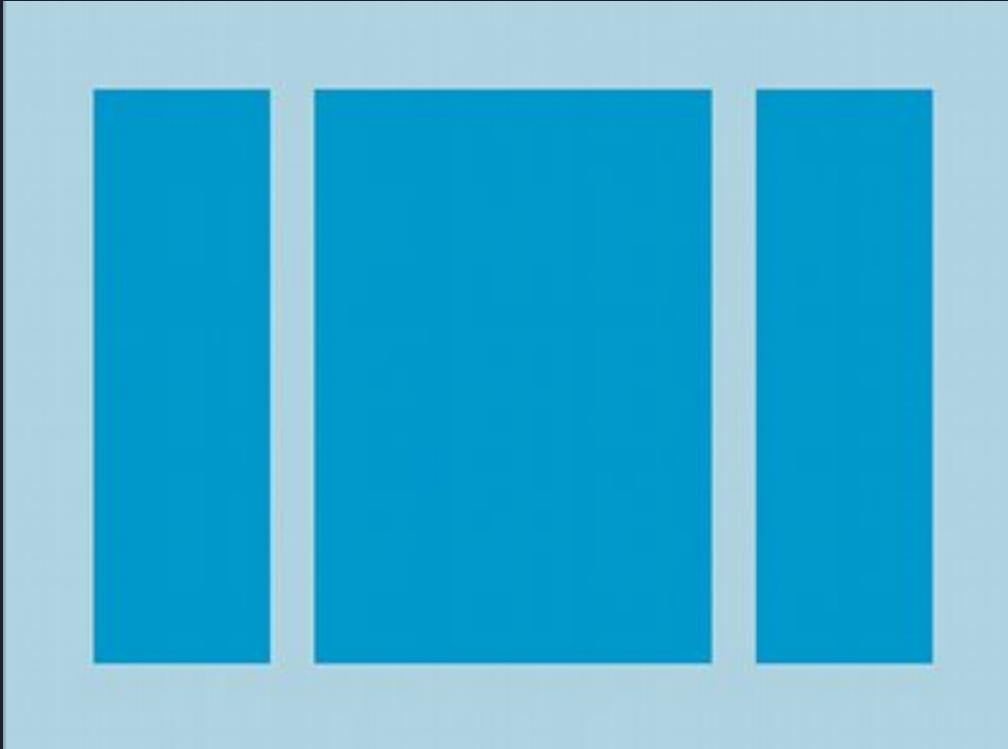
- Attributes**: Top-level section containing basic properties like `id`, `text`, and `style`.
- Declared Attributes**: Section under `button` containing `layout_width` and `text`. The `text` field is highlighted with a yellow border.
- Layout**: Section containing a `ConstraintWidget` diagram. The diagram shows the button's position relative to its parent view, with horizontal and vertical constraints set to 0dp.
- Constraints**: Section listing `Start` and `Top` constraints relative to the parent.
- Common Attributes**: Section containing properties like `style`, `stateListAnimator`, and `background`. The `text` field is highlighted with a yellow border.
- All Attributes**: Bottom-most section listing all available attributes for the button.

Numbered callouts point to specific elements:

- 1: Points to the `text` field in the `Declared Attributes` section.
- 2: Points to the `text` field in the `Common Attributes` section.
- 3: Points to the `text` field in the `All Attributes` section.
- 4: Points to the search bar at the top of the editor.
- 5: Points to the `layout_width` field in the `Declared Attributes` section.
- 6: Points to the `text` field in the `Declared Attributes` section.



Linear Layout



View group that aligns all children in a single direction, vertically or horizontally



Linear Layout

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >
    <TextView android:id="@+id/text"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello, I am a TextView" />
    <Button android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello, I am a Button" />
</LinearLayout>
```



Relative Layout



**View group that displays
child views in relative
positions**

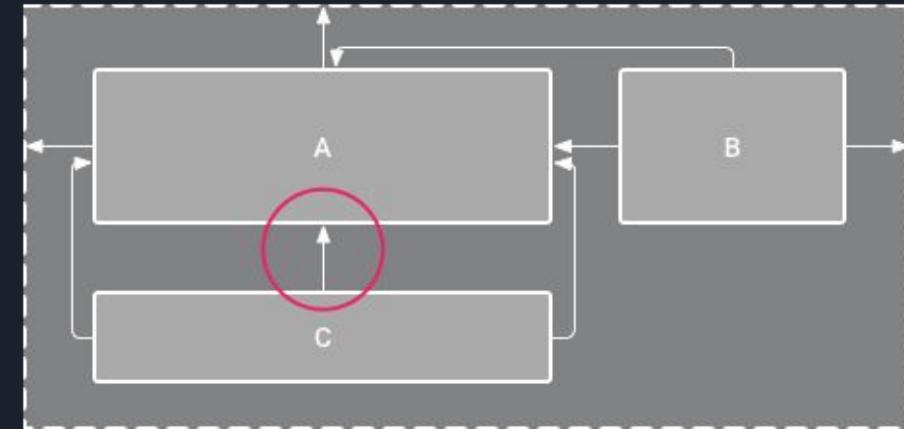
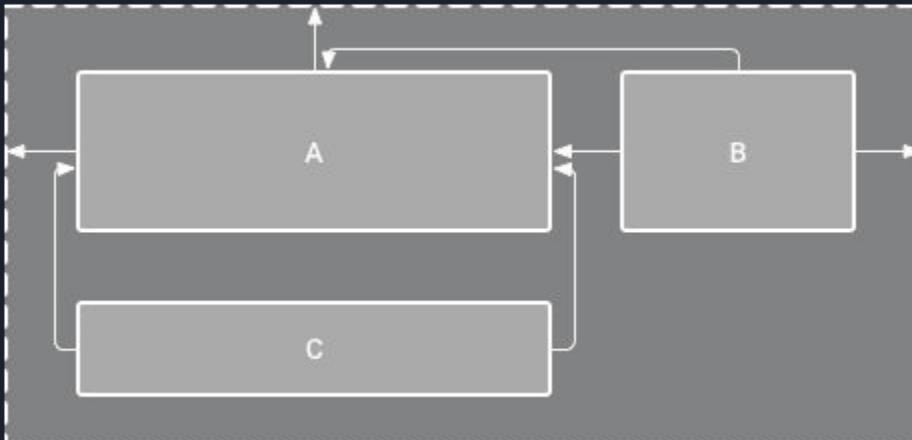


Relative Layout

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingLeft="16dp"
    android:paddingRight="16dp" >
    <EditText
        android:id="@+id/name"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="@string/reminder" />
    <Button
        android:layout_width="96dp"
        android:layout_height="wrap_content"
        android:layout_below="@id/name"
        android:layout_alignParentRight="true"
        android:text="@string/done" />
</RelativeLayout>
```



Constraint Layout



```
// build.gradle (app)  
dependencies {  
    implementation "androidx.constraintlayout:constraintlayout:2.1.0"  
}
```



Constraint Layout

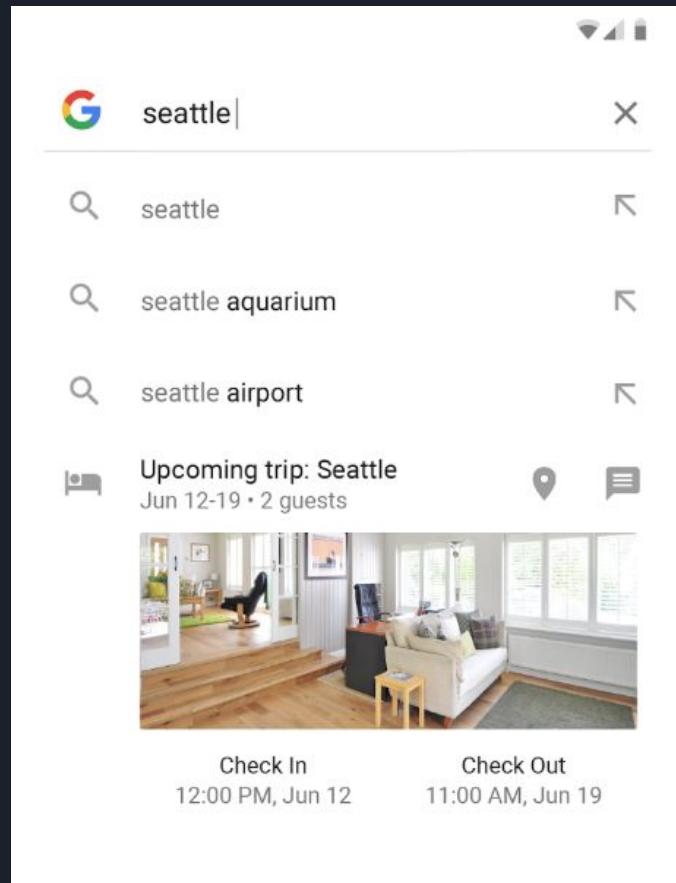
```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <Button
        android:id="@+id/button2"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:text="Button"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintBottom_toBottomOf="parent" />
</androidx.constraintlayout.widget.ConstraintLayout>
```



Recycler View

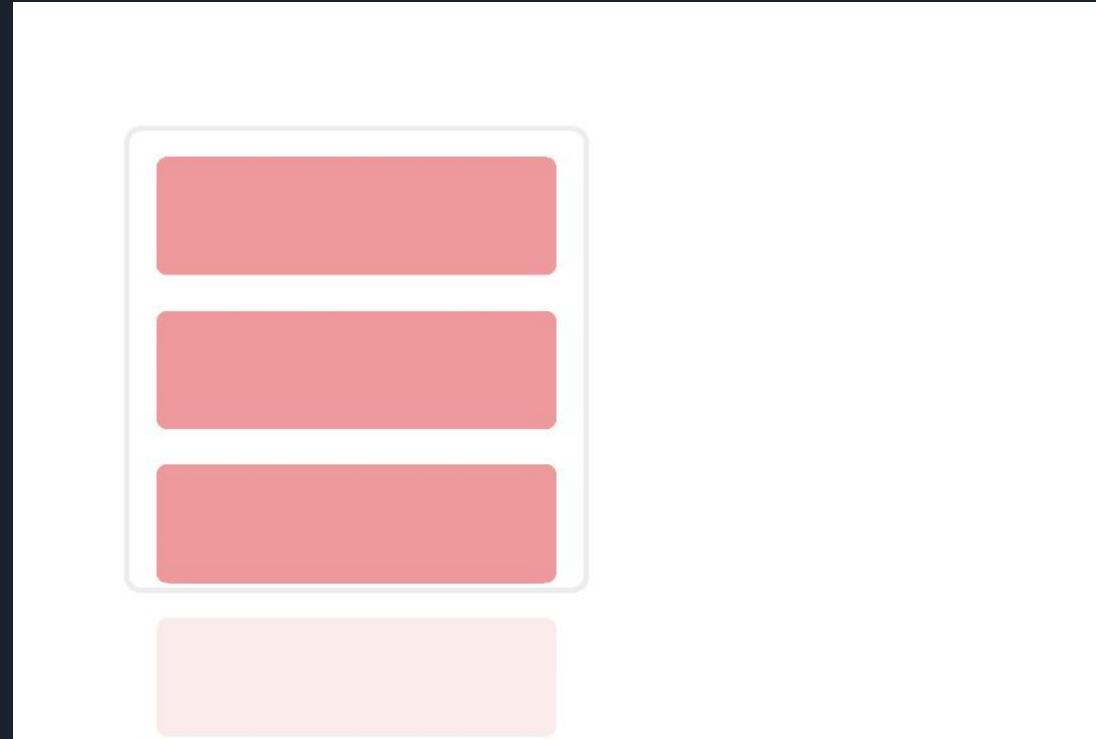
Display large sets of data in your UI while minimizing memory usage.





Recycler View

Display large sets of data in your UI while minimizing memory usage.



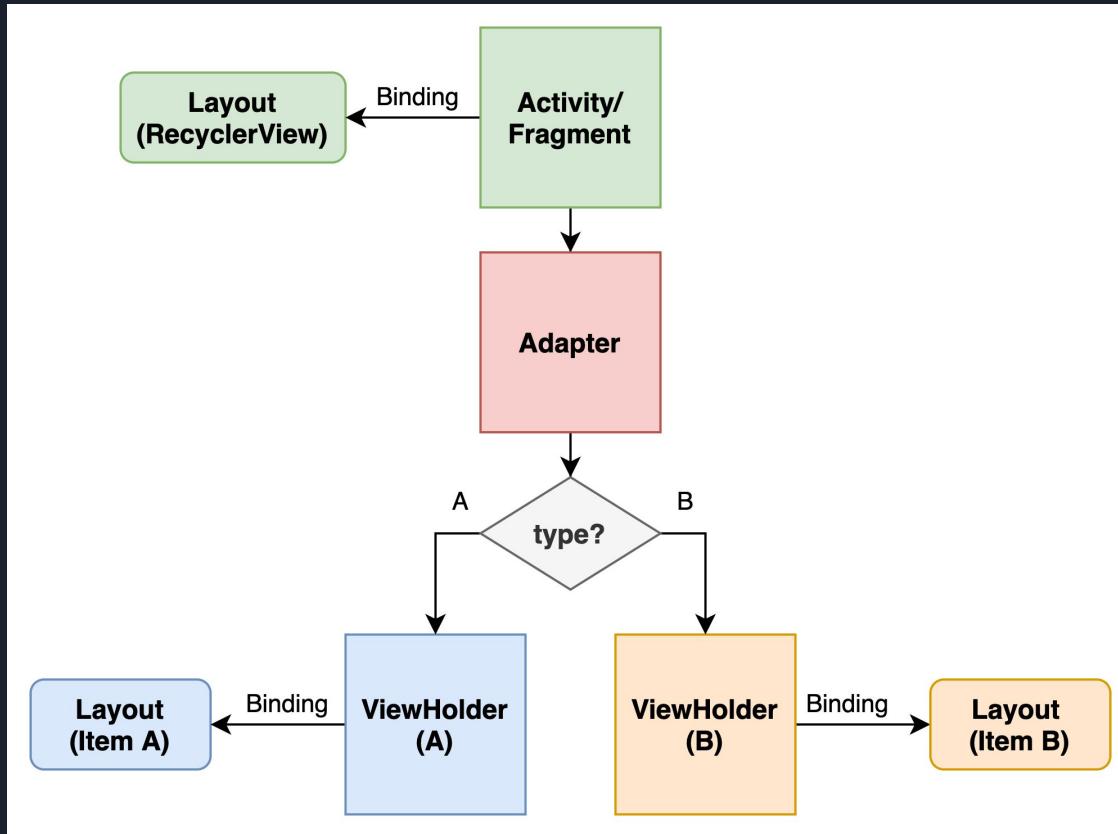


Recycler View

```
// build.gradle (app)  
dependencies {  
    implementation "androidx.recyclerview:recyclerview:1.2.1"  
}
```



Recycler View





View Binding

View binding is a feature that allows you to more easily write code that interacts with views. Once view binding is enabled in a module, it generates a binding class for each XML layout file present in that module. An instance of a binding class contains direct references to all views that have an ID in the corresponding layout.

In most cases, view binding replaces **findViewById**.



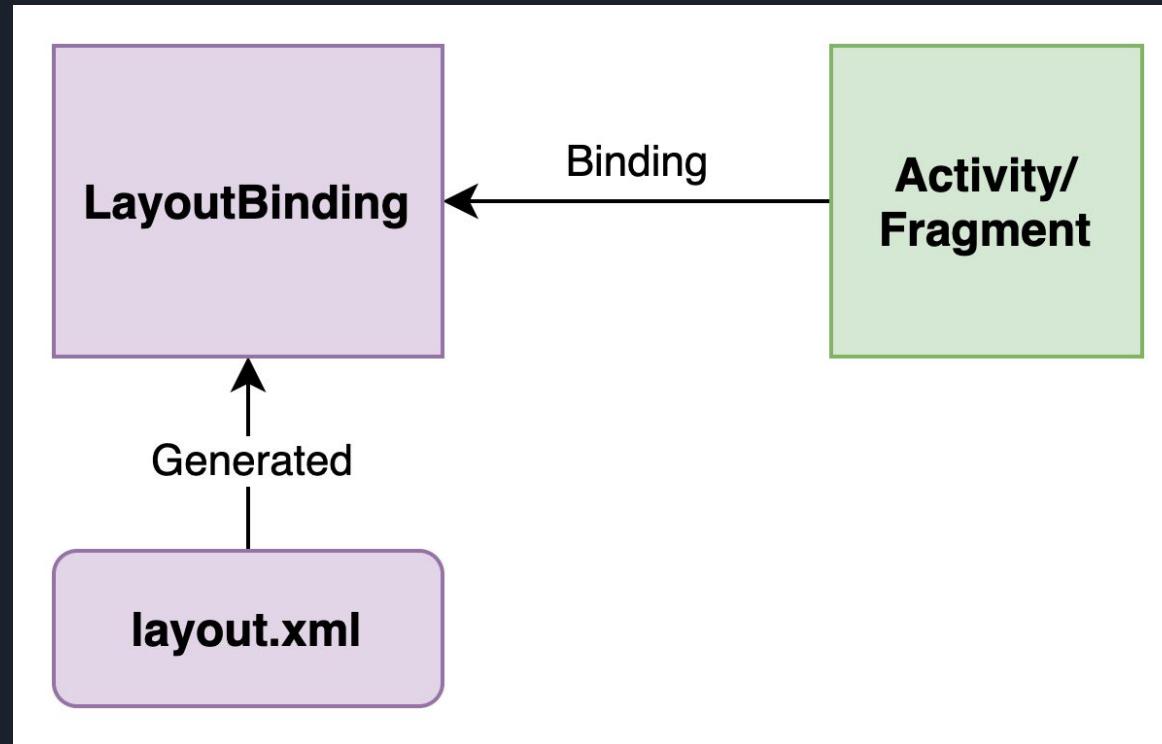
View Binding

```
// build.gradle (app)

android {
    buildFeatures {
        viewBinding true
    }
}
```



View Binding





Basic Kotlin

```
fun main() {  
    println("Hello world!")  
}
```



Basic Kotlin: Package definition and imports

```
package my.demo
```

```
import kotlin.text.*
```



Basic Kotlin: Variables

`var/val: Type = value`

Read-only local variables are defined using the keyword `val`. They can be assigned a value only once.

```
val a: Int = 1      // immediate assignment
val b = 2          // `Int` type is inferred
val c: Int          // Type required when no initializer is provided
c = 3              // deferred assignment
```

Variables that can be reassigned use the `var` keyword:

```
var x = 5          // `Int` type is inferred
x += 1
```

```
const val CONST_VALUE = "constant"
```



Basic Kotlin: Comments

```
// This is an end-of-line comment
```

```
/* This is a block comment  
on multiple lines. */
```



Basic Kotlin: String templates

```
var a = 1
// simple name in template:
val s1 = "a is $a"

a = 2
// arbitrary expression in template:
val s2 = "${s1.replace("is", "was")}, but now is $a"
println(s2) // Output: a was 1, but now is 2
```



Basic Kotlin: Control Flow: if

```
// Traditional usage
```

```
var max = a
if (a < b) max = b
```

```
// With else
```

```
var max: Int
if (a > b) {
    max = a
} else {
    max = b
}
```

```
// As expression
```

```
val max = if (a > b) a else b
```

* no ternary operator (*condition ? then : else*)



Basic Kotlin: Control Flow: when

```
val x = 1
when (x) {
    1 -> print("x == 1")
    2 -> print("x == 2")
    else -> { // Note the block
        print("x is neither 1 nor 2")
    }
}
```

```
val obj: Any = ""
when (obj) {
    1           -> "One"
    "Hello"     -> "Greeting"
    is Long      -> "Long"
    !is String   -> "Not a string"
    else         -> "Unknown"
}
```



Basic Kotlin: Control Flow: for

```
for (item in collection) print(item)
```

```
for (item: Int in ints) {  
    // ...  
}
```

```
for (i in 1..3) print(i)  
// 123
```

```
for (i in 1..8 step 2) print(i)  
// 1357
```

```
for (i in 6 downTo 0 step 2) {  
    print(i)  
}  
// 6420
```



```
(1..3).forEach {  
    print(it)  
}
```

* No this format like in java

```
for (int i = first; i <= last; i += step) {  
    // ...  
}
```



Basic Kotlin: Control Flow: while

```
while (x > 0) {  
    x--  
}
```

```
do {  
    val y = 0  
} while (y != 0) // y is visible here!
```



Basic Kotlin: Null Safety

```
var a: String = "abc"      // Regular initialization means non-null by default  
a = null                  // compilation error
```

```
var b: String? = "abc"    // can be set null  
b = null                 // ok
```

Safe Calls

```
val a = "Kotlin"  
val b: String? = null  
println(b?.length)  
println(a?.length)        // Unnecessary safe call
```



Basic Kotlin: Null Safety

Elvis Operator (?:)

```
val len: Int = if (b != null) b.length else -1      // Normal checking  
val len = b?.length ?: -1                          // Elvis Operator
```

The !! Operator (NPE-lovers)

```
val b: String? = null  
val len = b!!.length                                // beware to use
```

Safe Casts

```
val aInt: Int? = a as? Int
```



Basic Kotlin: Functions

```
val fruits = listOf("banana", "avocado", "apple", "kiwifruit")
fruits
    .filter { it.startsWith("a") }
    .sortedBy { it }
    .map { it.toUpperCase() }
    .forEach { println(it) }
```



Basic Kotlin: Functions

```
val fruits = listOf("banana", "avocado", "apple", "kiwifruit")
fruits
    .filter { it.startsWith("a") }
    .sortedBy { it }
    .map { it.toUpperCase() }
    .forEach { println(it) }
```

```
//APPLE
//AVOCADO
```



Basic Kotlin: Functions

// Function having two Int parameters with Int return type:

```
fun sum1(a: Int, b: Int): Int {  
    return a + b  
}
```

sum1(1, 2)

// call with alias name
sum1(a = 1, b = 2)

// Function with an expression body and inferred return type:

```
fun sum2(a: Int = 0, b: Int = 1) = a + b
```

Default value

// Function returning no meaningful value:

```
fun printSum1(a: Int, b: Int): Unit {  
    println("sum of $a and $b is ${a + b}")  
}
```

// Unit return type can be omitted:

```
fun printSum2(a: Int, b: Int) {  
    println("sum of $a and $b is ${a + b}")  
}
```



Basic Kotlin: Functions

```
// Generic functions
fun <T> someList(item: T): List<T> {
    /*...*/
}
```

```
// Extension functions
fun String.removeDash(): String {
    return replace("-", "")
}
```

```
val str = "12-34".removeDash()
println(str) // 1234
```



Basic Kotlin: Higher-Order Functions

```
/**  
 * Performs the given [action] on each element.  
 */  
public inline fun <T> Iterable<T>.forEach(action: (T) -> Unit): Unit {  
    for (element in this) action(element)  
}
```

Function types

```
(Int) -> String  
(() -> Unit)
```

```
public inline fun <T> Iterable<T>.forEachIndexed(action: (index: Int, T) -> Unit): Unit {  
    var index = 0  
    for (item in this) action(checkIndexOverflow(index++), item)  
}
```

```
val list = listOf(1, 2, 3)  
list.forEachIndexed { _, value -> println("$value!") }
```

_ parameter is unused



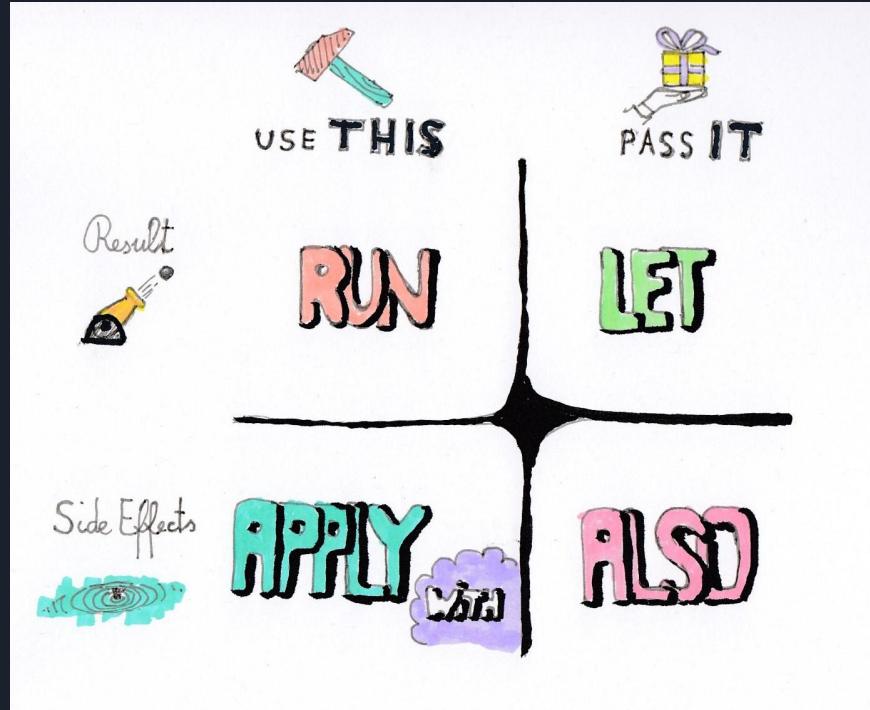
Basic Kotlin: Functions: Lambda expression

```
val sumFull: (Int, Int) -> Int = { x: Int, y: Int -> x + y }  
val sum = { x: Int, y: Int -> x + y }
```

```
print(sumFull(1, 2))  
print(sum(1, 2))
```



Basic Kotlin: Functions: Scope Function



<https://medium.com/@ramtop/kotlin-scope-functions-c8c41f0961f>

<https://kotlinlang.org/docs/scope-functions.html>



Basic Kotlin: Classes

```
interface Shapable {  
    fun area(): Double  
}  
  
open class Shape : Shapable {  
    open val vertexCount: Int = 0  
  
    open fun draw() {  
        println("draw on Shape $vertexCount")  
    }  
  
    fun fill() {  
        println("fill on Shape")  
    }  
  
    override fun area(): Double {  
        return 0.0  
    }  
}
```

private means visible inside this class only (including all its members);

protected – same as private + visible in subclasses too;

internal – any client inside this module who sees the declaring class sees its internal members;

public – any client who sees the declaring class sees its public members.

```
val shape = Shape()  
shape.draw()
```



Basic Kotlin: Classes

```
class Rectangle : Shape() {  
  
    override val vertexCount = 4  
  
    override fun draw() {  
        println("call super.draw() on Rectangle")  
        super.draw()  
    }  
}
```



Basic Kotlin: Classes

```
interface Circleable {  
    var radius: Double  
}  
  
class Circle : Shape(), Circleable {  
  
    override var radius: Double = 2.0  
  
    override fun draw() {  
        println("draw on Circle $vertexCount")  
    }  
  
    override fun area(): Double {  
        return PI * radius * radius  
    }  
}
```



Basic Kotlin: Classes

```
class ClassStatic {  
  
    val normalValue = "this is normal value"  
  
    companion object {  
        const val STATIC_VALUE = "this is static value"  
    }  
}  
  
object ClassObject {  
    const val STATIC_VALUE = "this is static value"  
}  
  
data class User(val name: String, val age: Int)
```

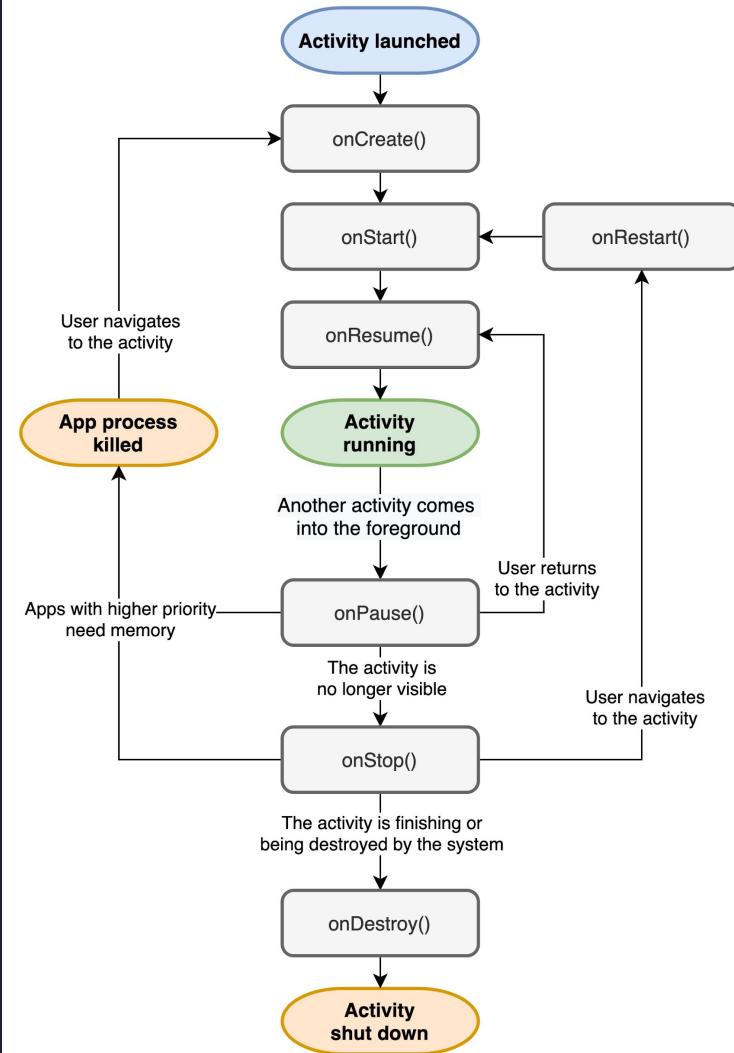


Activity

The **Activity** class is a crucial component of an Android app, and the way activities are launched and put together is a fundamental part of the platform's application model.

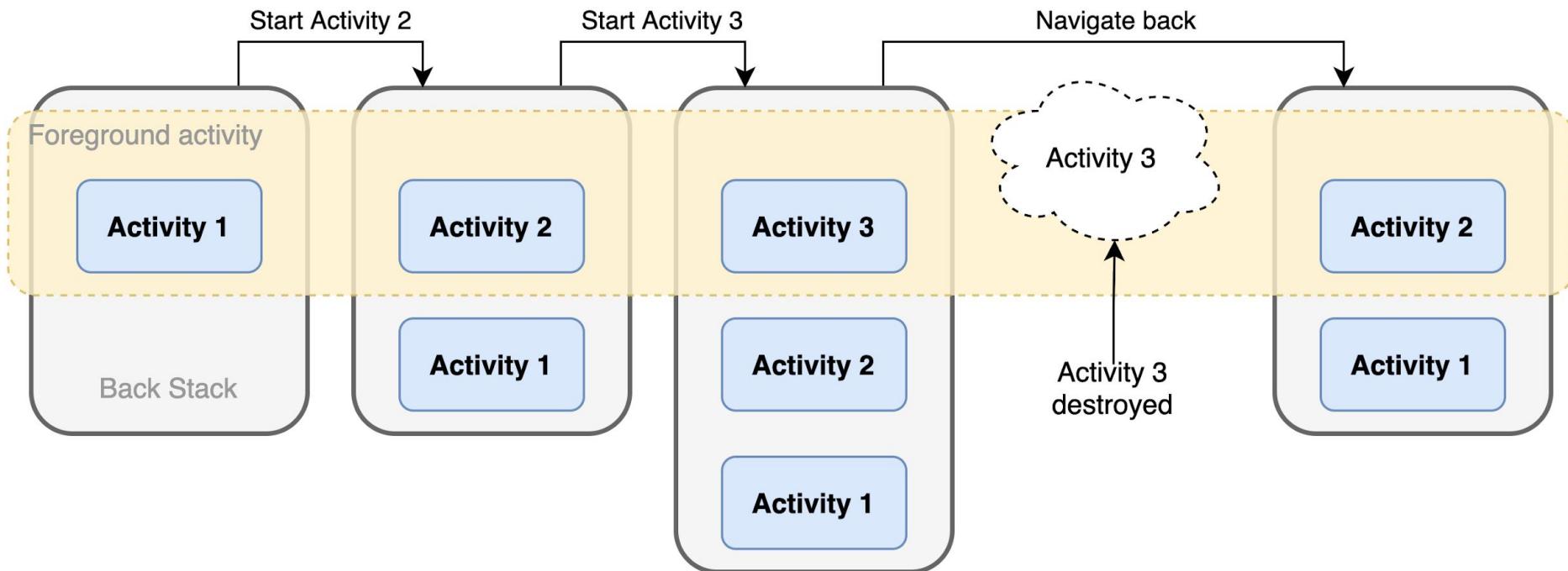


Activity: Lifecycle





Activity: Navigation



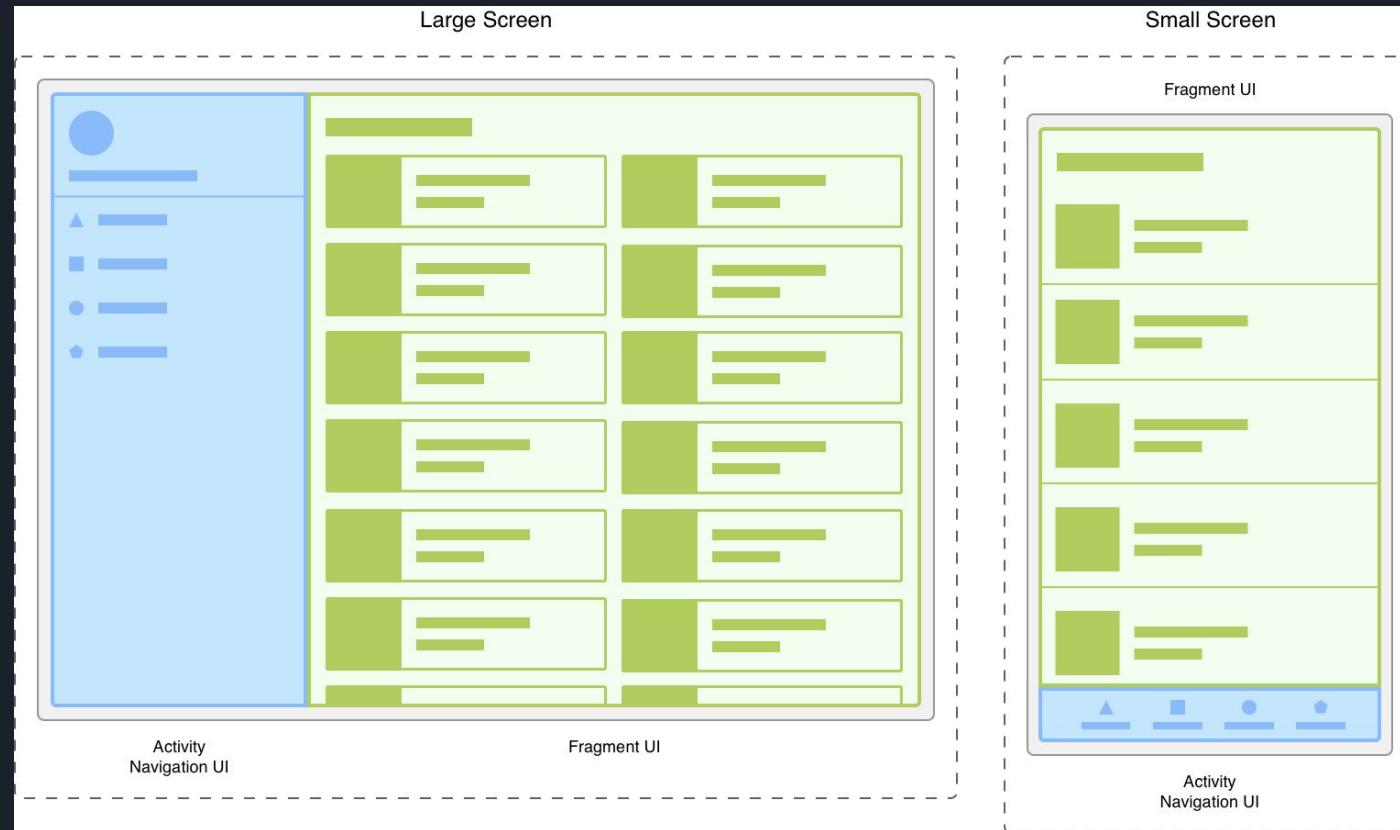


Fragment

A **Fragment** represents a reusable portion of your app's UI. A fragment defines and manages its own layout, has its own lifecycle, and can handle its own input events. Fragments cannot live on their own--they must be hosted by an activity or another fragment. The fragment's view hierarchy becomes part of, or attaches to, the host's view hierarchy.

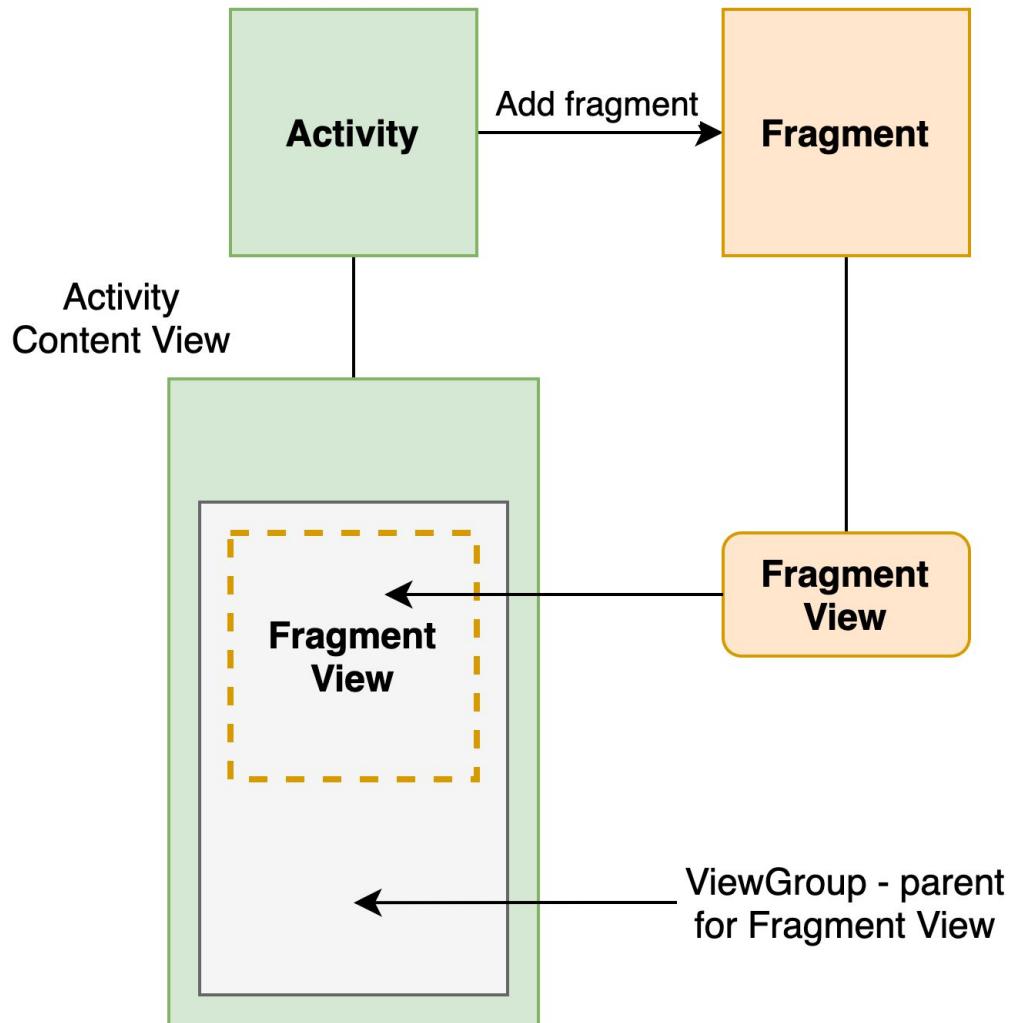


Fragment





Fragment



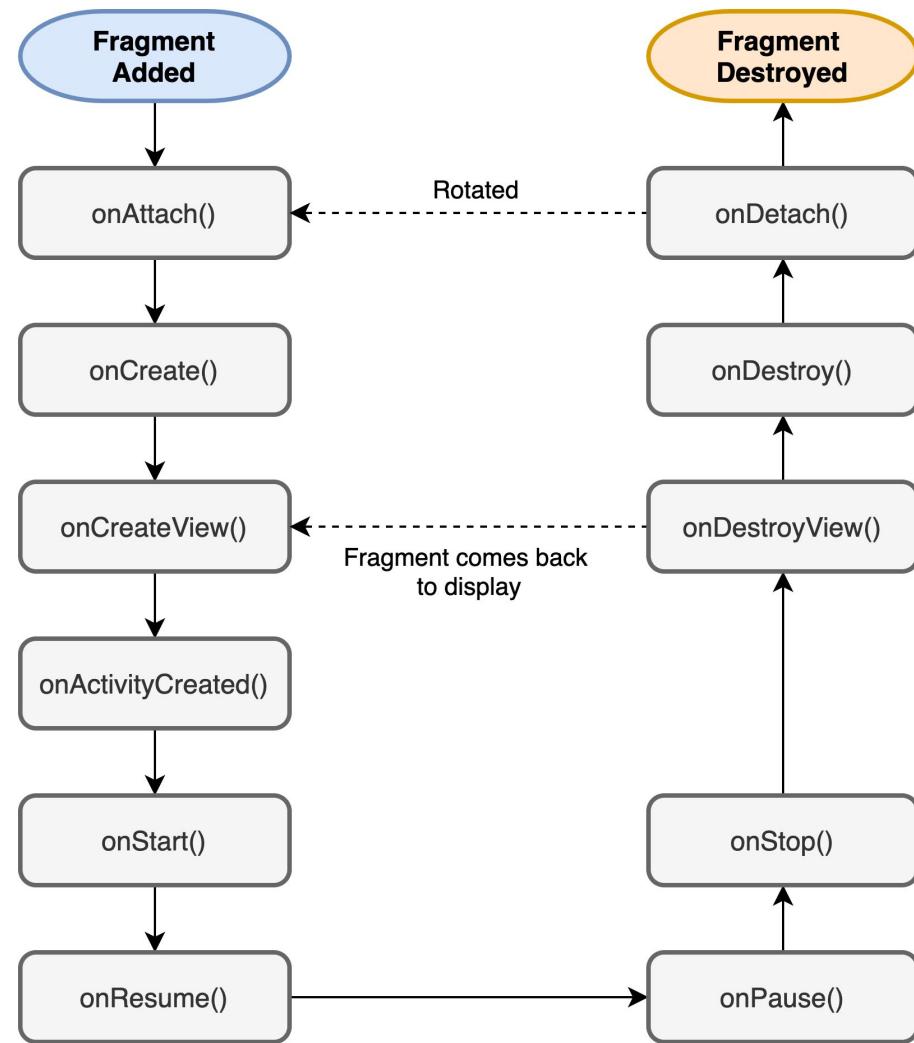


Fragment

```
// build.gradle (app)
dependencies {
    def fragment_version = "1.3.6"
    implementation "androidx.fragment:fragment-ktx:$fragment_version"
}
```

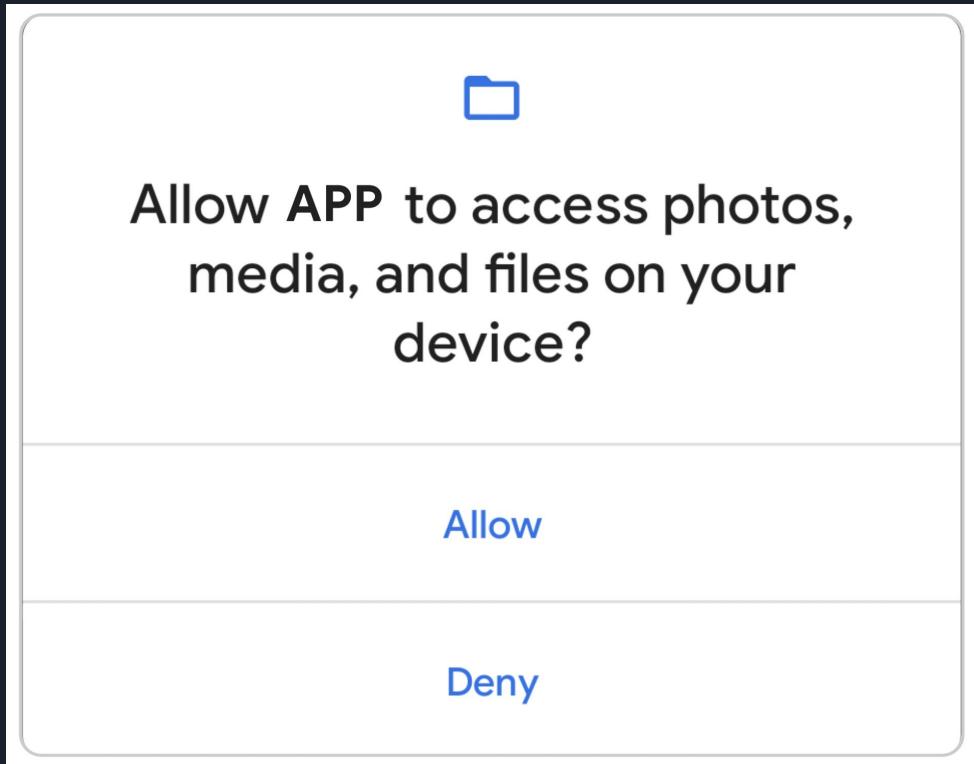


Fragment: Lifecycle



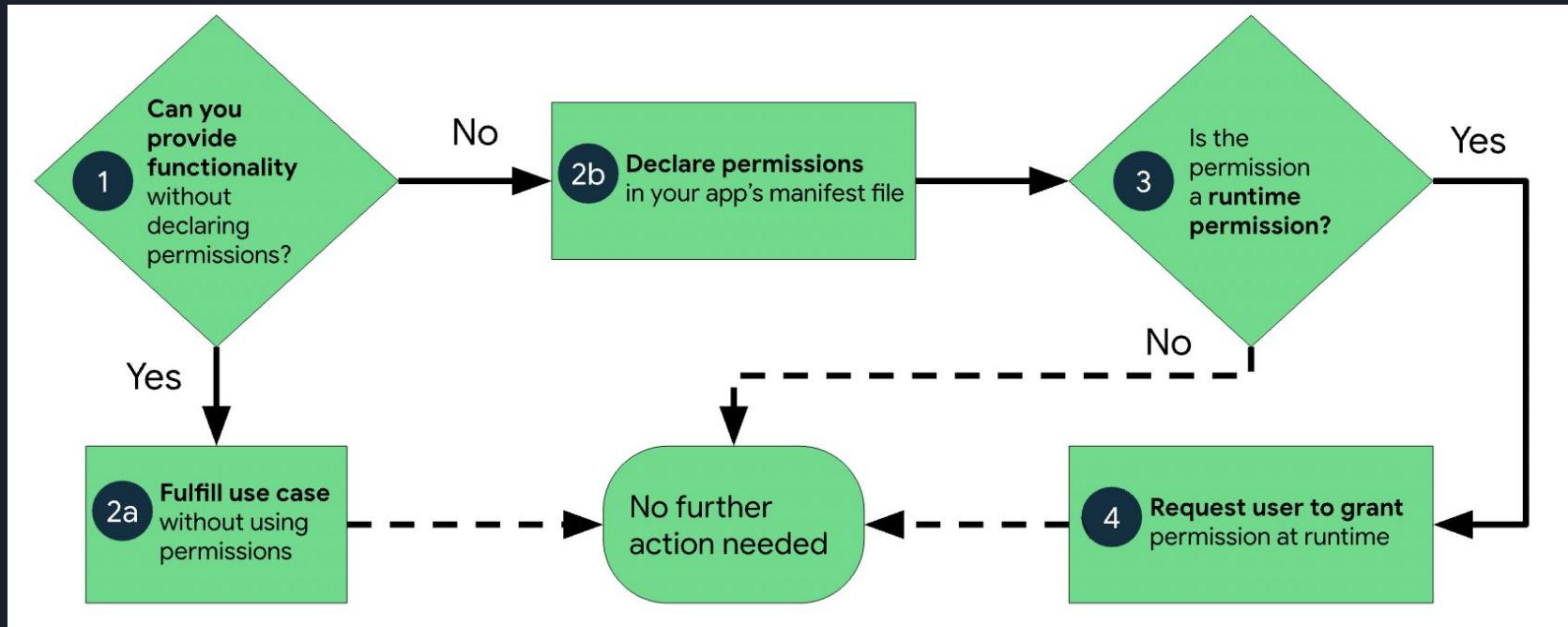


Permissions



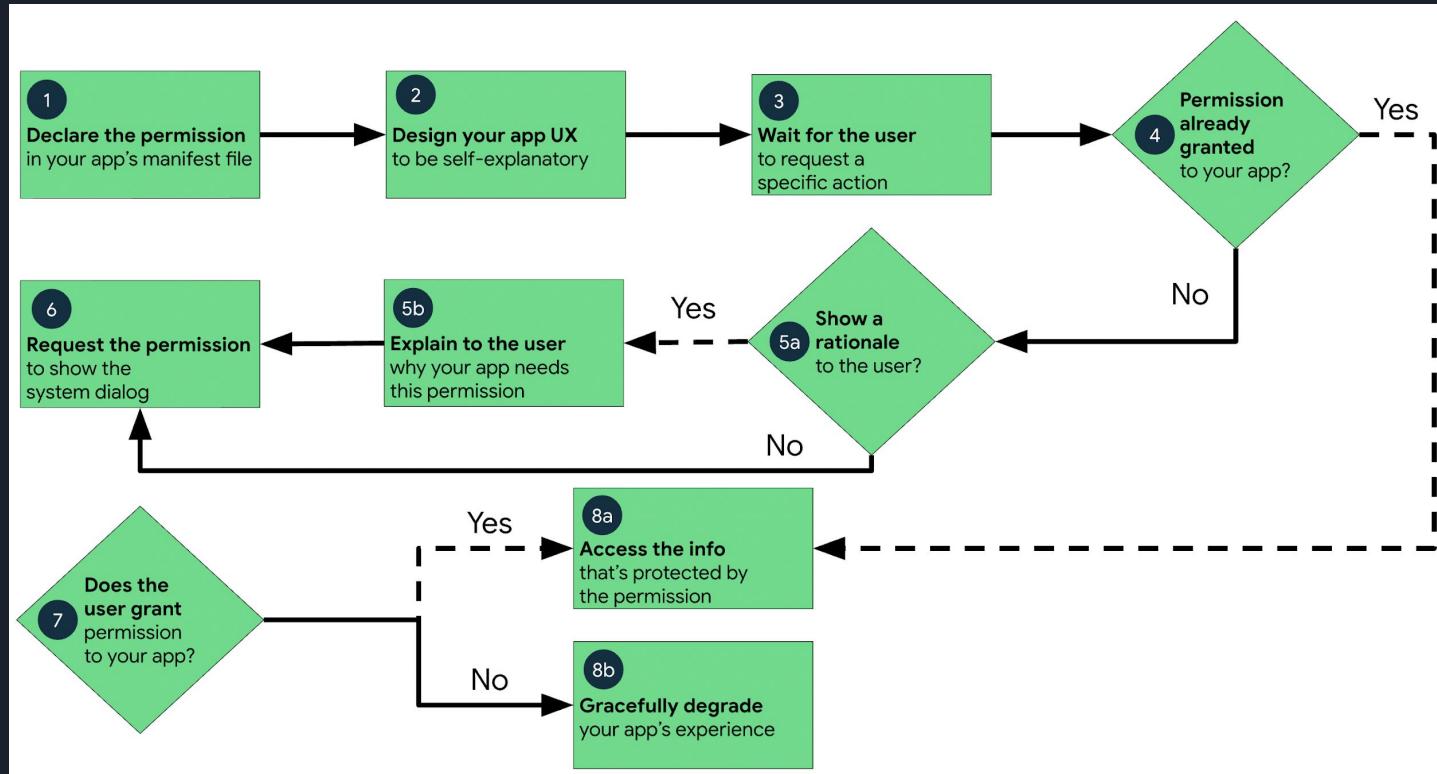


Permissions





Permissions





Permissions

```
// AndroidManifest.xml

<manifest ...>
    <!-- Add declaration to app manifest -->
    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.CAMERA" />

    <!-- Declare hardware as optional -->
    <uses-feature android:name="android.hardware.camera"
                  android:required="false" />

    <application ...>
        ...
    </application>
</manifest>
```



Permissions

Dexter

<https://github.com/Karumi/Dexter>

PermissionsDispatcher

<https://github.com/permissions-dispatcher/PermissionsDispatcher>

RxPermissions

<https://github.com/tbruyelle/RxPermissions>

<https://developer.android.com/training/permissions/requesting>



Android Jetpack



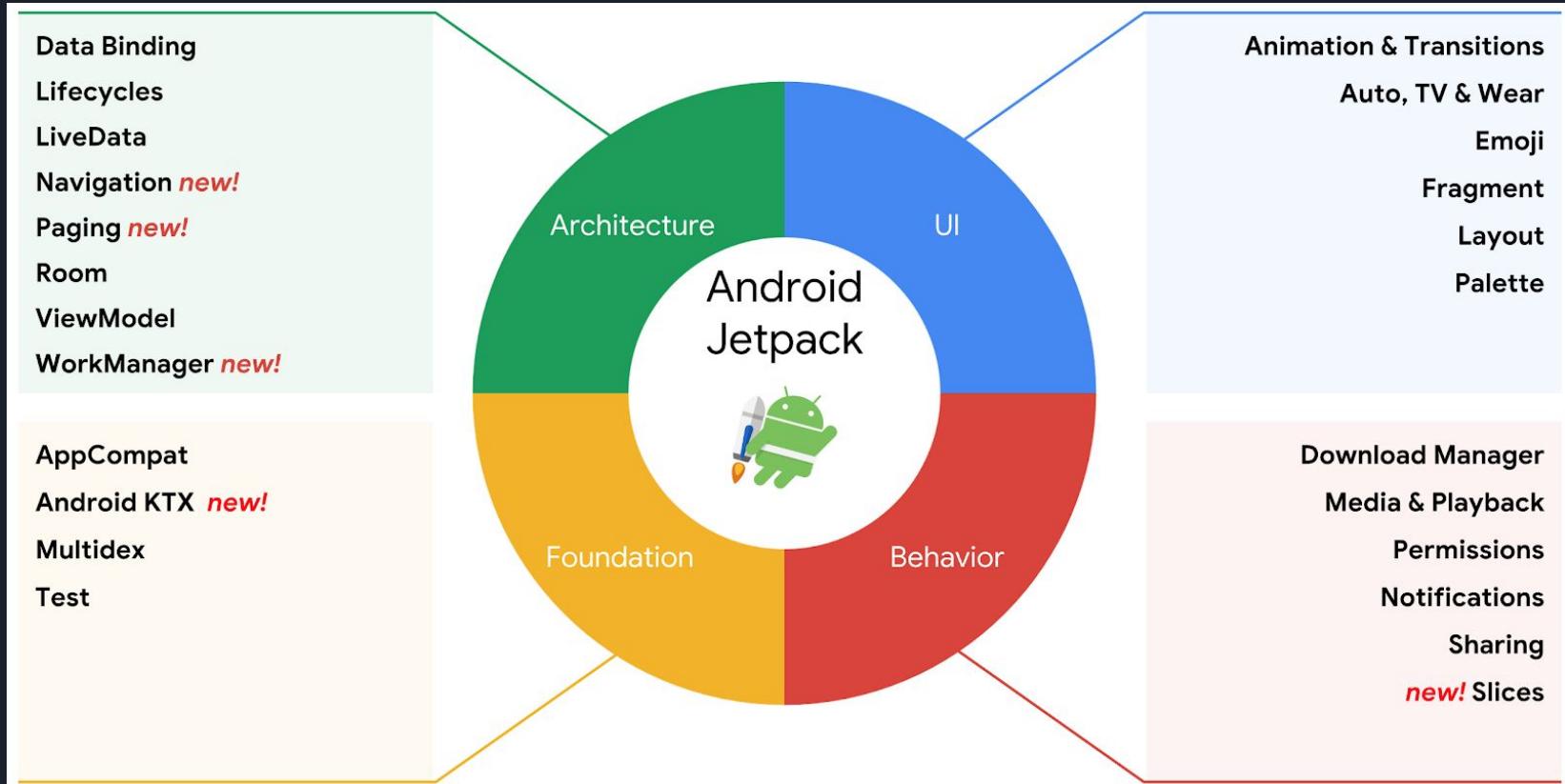


Android Jetpack

Jetpack is a suite of libraries to help developers follow best practices, reduce boilerplate code, and write code that works consistently across Android versions and devices so that developers can focus on the code they care about.



Android Jetpack





Jetpack Compose





Jetpack Compose

Jetpack Compose is Android's modern toolkit for building native UI. It simplifies and accelerates UI development on Android. Quickly bring your app to life with less code, powerful tools, and intuitive Kotlin APIs.



Navigation Component

nav_graph.xml fragment_view_balance.xml fragment_main.xml

Destinations navigation

HOST —

activity_main (fragment)

GRAPH —

fragmentMain - Start
fragmentViewBalance
navigation

navigation

Nested Graph

fragmentMain

VIEW TRANSACTIONS
VIEW BALANCE
SEND MONEY

fragmentViewBalance...

\$1

Attributes

nav_graph navigation

id nav_graph

label

startDestination fragmentMain

► Argument Default Values

► Global Actions

► Deep Links

1 2 3

```
graph TD; fragmentMain --> fragmentViewBalance[fragmentViewBalance...]
```



Navigation Component

The **Navigation component** consists of three key parts that are described below:

- **Navigation graph:** An XML resource that contains all navigation-related information in one centralized location. This includes all of the individual content areas within your app, called *destinations*, as well as the possible paths that a user can take through your app.
- **NavHost:** An empty container that displays destinations from your navigation graph. The Navigation component contains a default NavHost implementation, **NavHostFragment**, that displays fragment destinations.
- **NavController:** An object that manages app navigation within a NavHost. The NavController orchestrates the swapping of destination content in the NavHost as users move throughout your app.



Data Binding

The Data Binding Library is a support library that allows you to bind UI components in your layouts to data sources in your app using a declarative format rather than programmatically.



Data Binding

```
// build.gradle (app)

android {
    buildFeatures {
        dataBinding true
    }
}
```



Data Binding

```
<layout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto">
    <data>
        <variable
            name="viewmodel"
            type="com.myapp.data.ViewModel" />
    </data>

    <ConstraintLayout... >
        <TextView
            android:text="@{viewmodel.userName}" />
    </ConstraintLayout>
</layout>
```



Data Binding

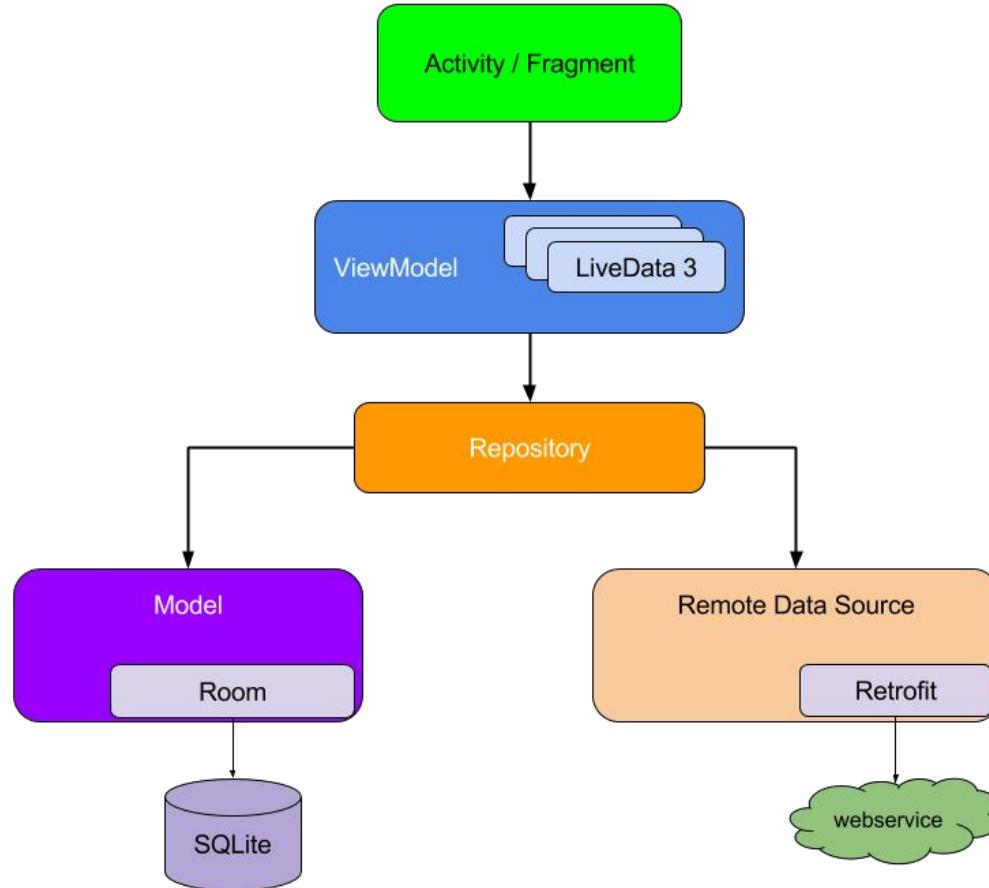
Two-way data binding:

<https://developer.android.com/topic/libraries/data-binding/two-way>



Android Architecture

Android architecture components are a collection of libraries that help you design robust, testable, and maintainable apps. Start with classes for managing your UI component lifecycle and handling data persistence.





LiveData

LiveData is an 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. This awareness ensures LiveData only updates app component observers that are in an active lifecycle state.



LiveData

```
// build.gradle (app)
dependencies {
    implementation "androidx.lifecycle:lifecycle-livedata-ktx:2.3.1"
}
```



ViewModel

The ViewModel class is 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.

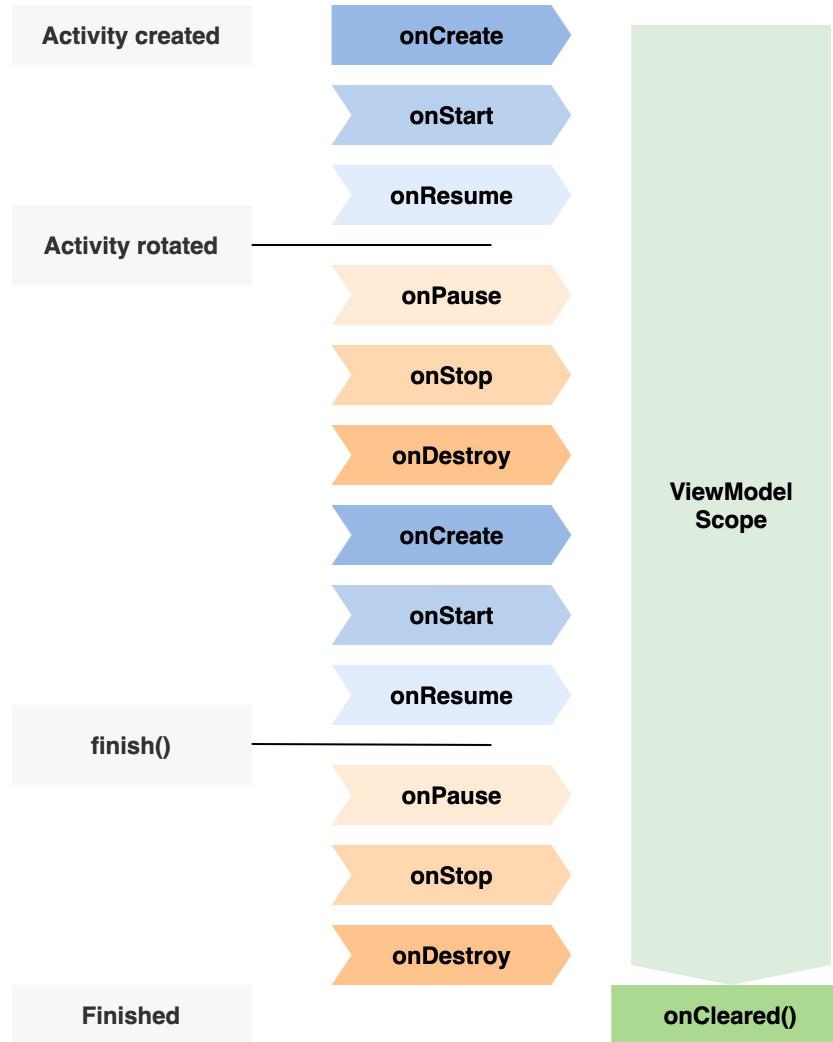


ViewModel

```
// build.gradle (app)
dependencies {
    implementation "androidx.lifecycle:lifecycle-viewmodel-ktx:2.3.1"
}
```



ViewModel





Network

- **OkHttp**: <https://square.github.io/okhttp/>

OkHttp is an HTTP client that's efficient by default

- **Retrofit**: <https://square.github.io/retrofit/>

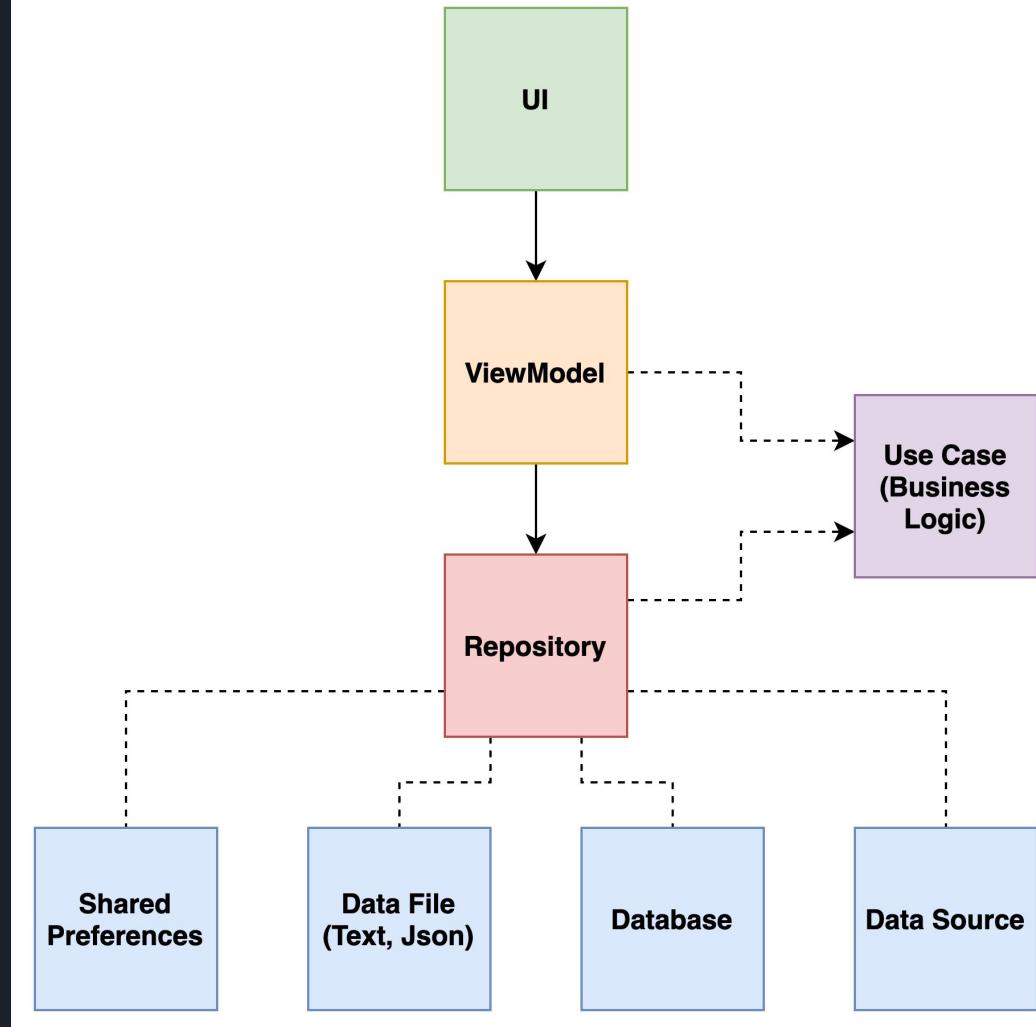
A type-safe HTTP client for Android and Java

- **Gson**: <https://github.com/google/gson>

Gson is a Java library that can be used to convert Java Objects into their JSON representation. It can also be used to convert a JSON string to an equivalent Java object. Gson can work with arbitrary Java objects including pre-existing objects that you do not have source-code of.

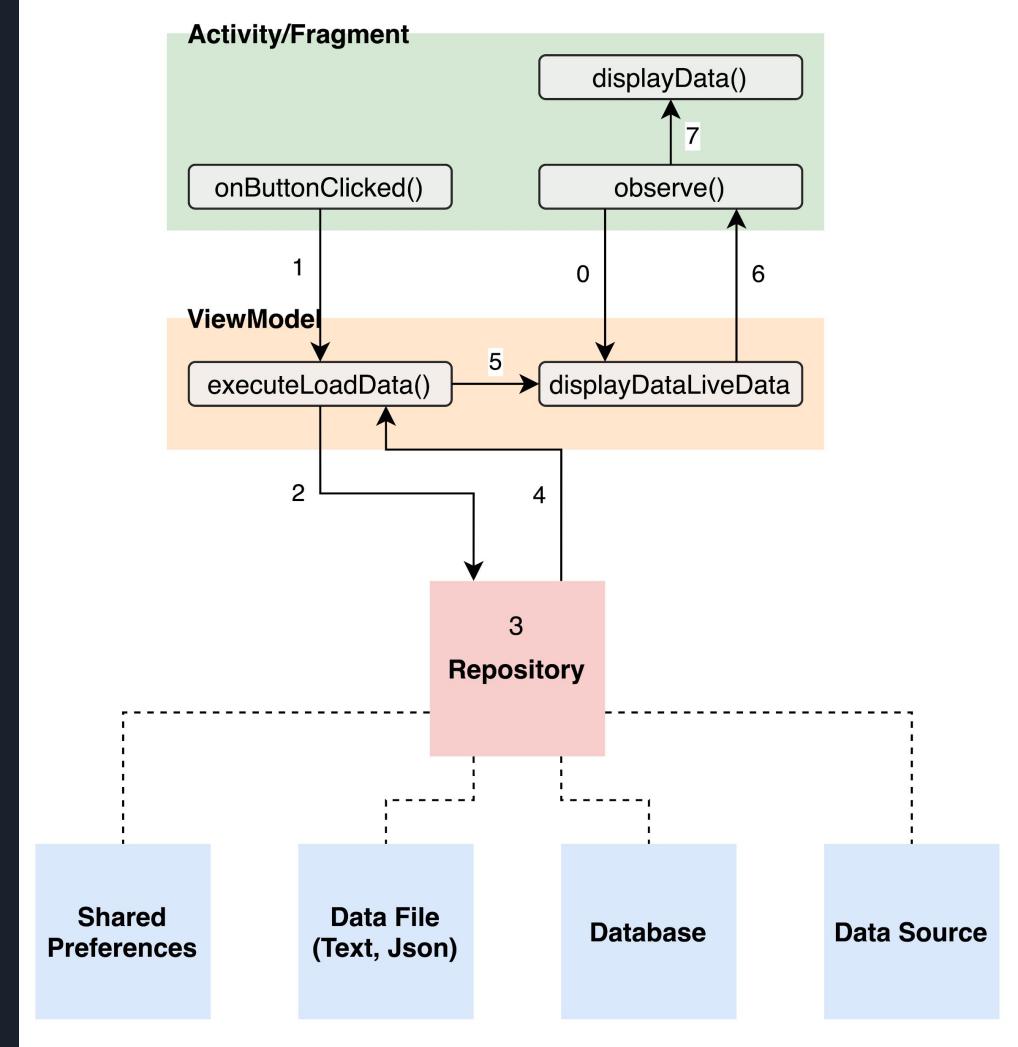


Structure workflow



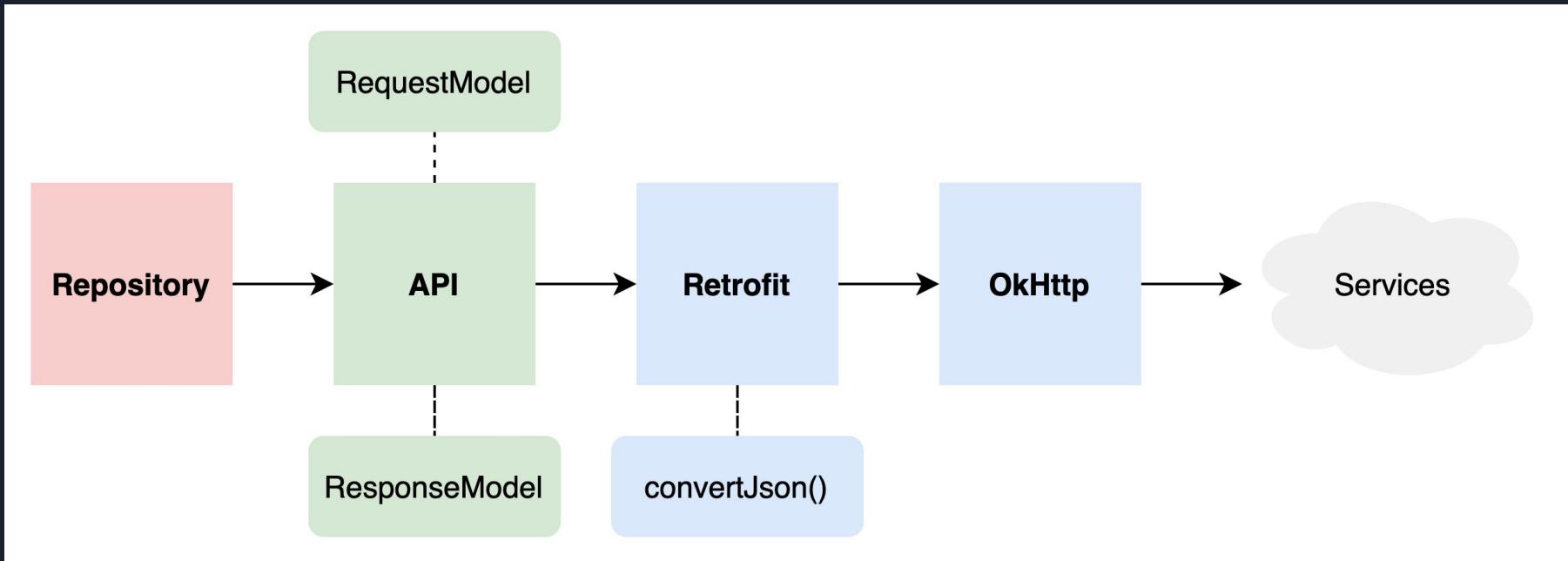


Structure workflow



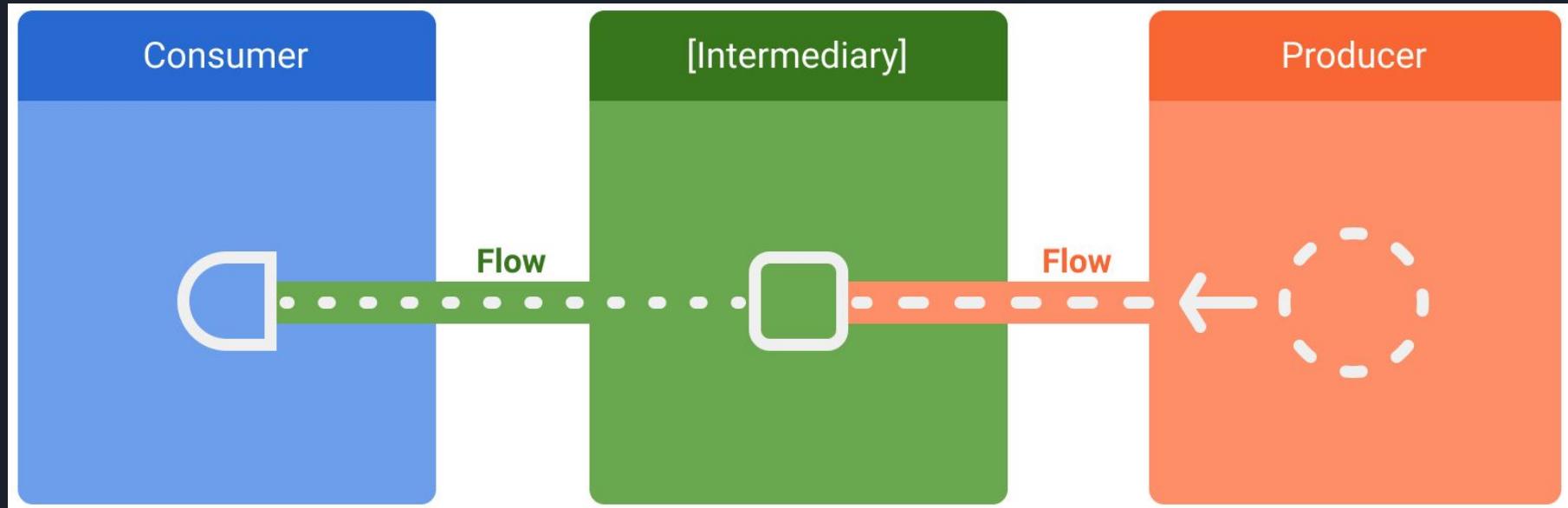


Structure workflow





Kotlin: Coroutines Flows





Kotlin: Coroutines Flows

A Flow is an **async sequence of values**

Flow produces values one at a time (instead of all at once) that can generate values from async operations like network requests, database calls, or other async code. It supports coroutines throughout its API, so you can transform a flow using coroutines as well!

Flow can be used in a fully-reactive programming style. If you've used something like RxJava before, Flow provides similar functionality. Application logic can be expressed succinctly by transforming a flow with functional operators such as [map](#), [flatMapLatest](#), [combine](#), and so on.



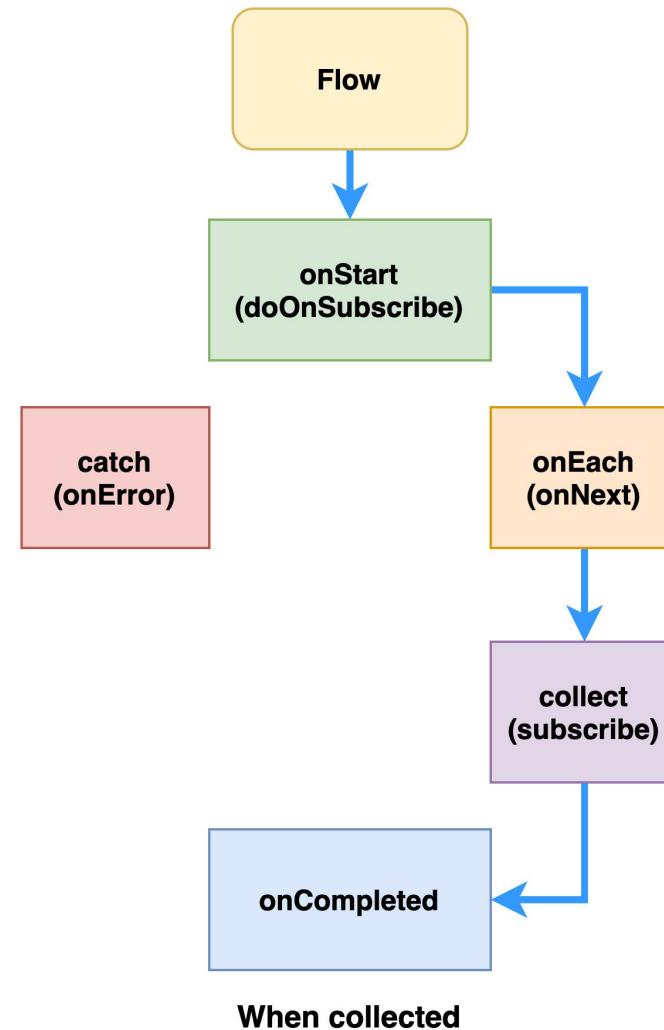
Kotlin: Coroutines Flows

```
// SomeViewModel.kt

fun executeGetData() {
    viewModelScope.launch {
        someRepository.getData()
            .onStart {
                showLoading()
            }
            .onCompletion {
                hideLoading()
            }
            .catch {
                showErrorDialog(it)
            }
            .collect {
                _displayData.postValue(it)
            }
    }
}
```

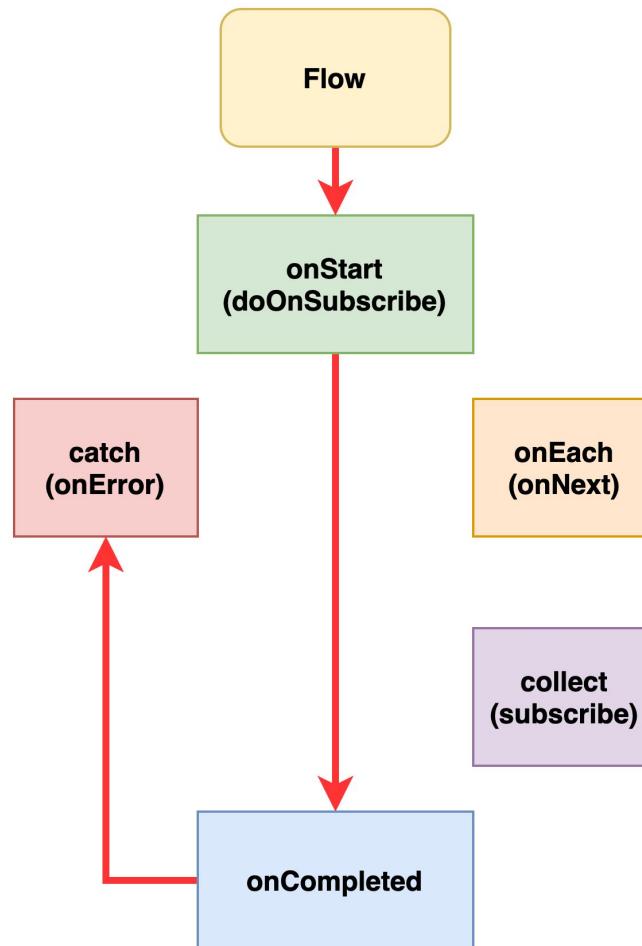


Kotlin: Coroutines Flows





Kotlin: Coroutines Flows



When throw some exception



Save key-value data

A **SharedPreferences** object points to a file containing key-value pairs and provides simple methods to read and write them. Each **SharedPreferences** file is managed by the framework and can be private or shared.

Work with data more securely:

<https://developer.android.com/topic/security/data>



Dependency Injection

Dependency injection (DI) is a technique widely used in programming and well suited to Android development. By following the principles of DI, you lay the groundwork for good app architecture.

Implementing dependency injection provides you with the following advantages:

- Reusability of code
- Ease of refactoring
- Ease of testing



Dependency Injection: Without

Car

Engine

```
class Car {  
  
    private val engine = Engine()  
  
    fun start() {  
        engine.start()  
    }  
  
}  
  
fun main(args: Array) {  
    val car = Car()  
    car.start()  
}
```



Dependency Injection

Engine



Car

```
class Car(private val engine: Engine){  
    fun start(){  
        engine.start()  
    }  
  
    fun main(args: Array){  
        val engine = Engine()  
        val car = Car(engine)  
        car.start()  
    }  
}
```

```
class Car {  
    lateinit var engine: Engine  
  
    fun start(){  
        engine.start()  
    }  
  
    fun main(args: Array){  
        val car = Car()  
        car.engine = Engine()  
        car.start()  
    }  
}
```



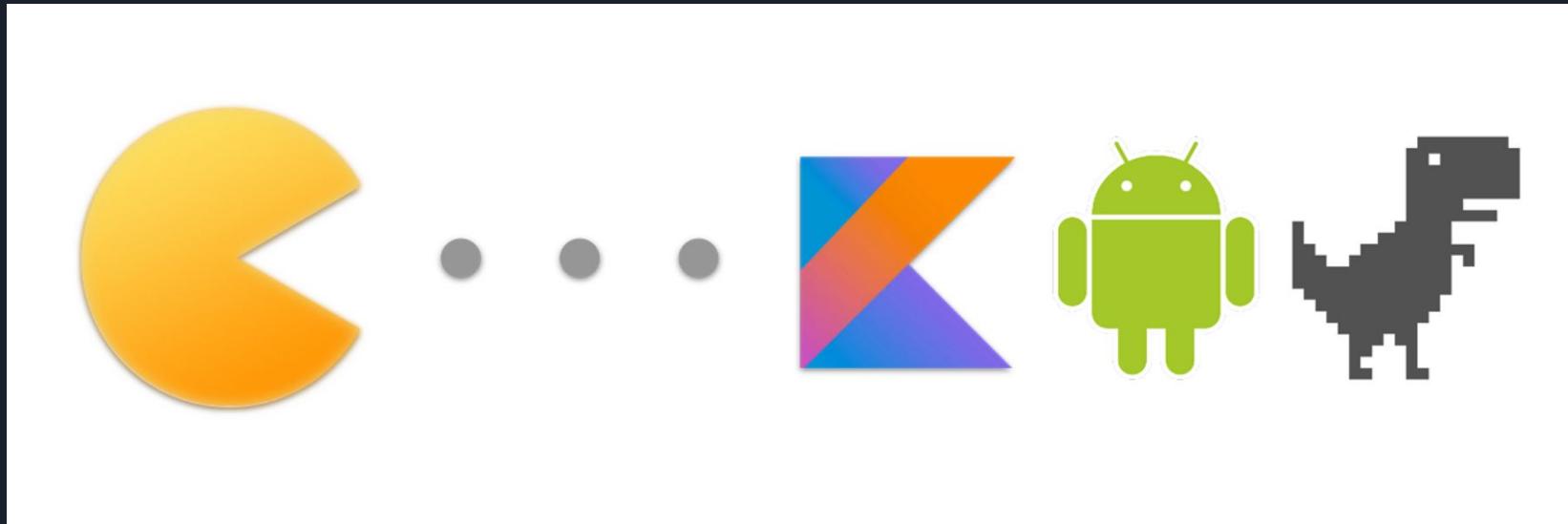
Dependency Injection: Hilt



**Hilt—Dependency Injection
Library for Android**



Dependency Injection: Koin





Mock Response



[Stubby4j](#)